



NEAR EAST UNIVERSITY

**GRADUATE SCHOOL OF
APPLIED AND SOCIAL SCIENCES**

**RECURRENT FUZZY NEURAL NETWORK BASED
MODELING AND CONTROL OF BATTERIES
CHARGING PROCESS**

Kaan Uyar

Ph.D. Thesis

Department of Computer Engineering

Nicosia - 2006


Kaan Uyar: Recurrent Fuzzy Neural Network Trained Modeling and Control of Batteries Charging Process


**Approval of Director of the
Graduate School of Applied and Social Sciences**


Prof. Dr Fakhreddin Mamedov
Director


We certify that this thesis is satisfactory for the award of the degree of Doctor of Philosophy in Computer Engineering


Examining Committee in Charge


Prof. Dr Rafik A. Aliev, Department of Computer-aided Control Systems,
Azerbaijan State Oil Academy, Azerbaijan


Prof. Dr Perviz Ali-zade, Department of Electrical Engineering, Istanbul
Technical University, Turkey


Prof. Dr Şenol Bektaş, Department of Electrical and Electronic Engineering,
Vice Rector, Near East University, TRNC


Prof. Dr Fakhreddin Mamedov, Dean, Faculty of Engineering, Vice Rector, Near
East University, TRNC


Prof. Dr İsmail Burhan Türkşen, Chairman, Department of Industrial
Engineering, TOBB - Economy and Technology University, Turkey

ACKNOWLEDGEMENT

"First, I would like to thanks to Kamile, my lovely wife, your support through this adventure kept my life balanced. I could not have completed this work without the encouragement of my mother Şaziye. Thank you both for all of your sacrifices. I must thank to my father Göral Mustafa for everything. Şebnem Ece and Göral Kaan, my dear little children, I love both of you so much.

I would like to thank the many faculty and staff who have enriched my experience at Near East University, specially the founder of the university Dr Suat İ. Günsel and President Prof. Dr Ümit Hassan, Vice-Presidents Prof. Dr Şenol Bektaş and Prof. Dr Fakhreddin Mamedov.

Finally, thanks to Babek Guirimov, Ümit İlhan and Okan Donangil for being patient study buddies."

ABSTRACT

Research works on intelligent chargers have received considerable attention in recent literature. The Nickel Cadmium (NiCd) battery charging is a nonlinear electrochemical dynamic process which has a high degree of uncertainty and lacks an exact mathematical model. There are several research works on applications of emerging technologies such as fuzzy, neural, genetic and neuro-fuzzy for battery charging. Unfortunately, progress in developing intelligent controller systems for NiCd battery charging has been limited, where existing chargers optimize charging time or battery temperature. Current intelligent battery chargers do not detect whether the battery is deeply discharged or shorted cell which can blow the battery if forced to charge. Thus, there is a need for an intelligent charger that optimizes both charging time and temperature while detecting the difference between deeply discharged and shorted cell before starting to charge the battery.

This thesis proposes a novel neuro fuzzy genetic approach to model and control NiCd battery charging process. The dynamics of NiCd battery are described by recurrent fuzzy neural networks (RFNN) where fuzzy control rules are generated. In addition, a new dynamic data mining technique for battery charging rules extracting is also suggested within this work. The simulation results of the proposed approach show more efficiency in comparison with existing intelligent chargers.

TABLE OF CONTENTS

APPROVAL	ii
ACKNOWLEDGEMENT	iii
ABSTRACT	iv
TABLE OF CONTENTS	v
LIST OF ABBREVIATIONS	viii
LIST OF TABLES	ix
LIST OF FIGURES	x
1. INTRODUCTION	1
2. STATE OF THE ART INTELLIGENT BATTERY CHARGING SYSTEMS	
2.1 Batteries as Control Object	3
2.1.1 Battery Chemistries	4
2.1.2 The Nickel Cadmium Battery	7
2.1.3 Charging the Nickel Cadmium Battery	12
2.2 Review of Existing Works of Intelligent Batteries Chargers	19
2.3 Statement of Research Problem	25
3. ARCHITECTURE OF NEURO-FUZZY-GENETIC CONTROL SYSTEM FOR BATTERIES CHARGING	
3.1 Structure of Control System and Description of its Working Principles	27
3.2 Elements of Batteries Charging Control Systems	28
4. RFNN AND THEIR LEARNING	
4.1 Review on FNN	37

4.2	Description of Investigated RFNN	38
4.3	GA based learning of RFNN	39
5.	FUZZY MODELING OF THE NiCd BATTERIES BY USING RFNN	
5.1	Approximation of the Nonlinear Dynamics of the NiCd Battery by RFNN	45
5.2	Investigation of Accuracy Neuro Fuzzy Model of NiCd Battery	49
5.3	Software Development Neuro-Fuzzy-Genetic Modeling of NiCd Batteries Charging	51
6.	EXPERIMENTAL INVESTIGATION OF INTELLIGENT NEURO-FUZZY-GENETIC CONTROL SYSTEM FOR NiCd BATTERY	
6.1	Dynamic data mining technique for battery charging rules extraction	53
6.2	Software Development NFG Control System for NiCd Batteries Charging	60
6.3	Experimental Results of Battery Control System	60
6.4	Compatible Analysis of NFG Control System for Batteries Charging	68
6.5	Identification of the Proposed Controller for NiCd Batteries Charger	69
	CONCLUSION	72
	REFERENCES	73
	APPENDIX	
A.	Programs	
A.1	Network_D.cs	78
A.2	Controller.cs	92

A.3	Charging_control.cs	97
A.4	FR_ChargingControl.cs	99
A.5	TrainRules.cs	105
A.6	Genetic_D.cs	105
A.7	Battery.cs	117
A.8	Macro of the NSIM-tem.xls file	121
A.9	Macro of the NSIM-vol.xls	123
A.10	sdkvars.bat	126
A.11	battery_dll.bat	126
A.12	TrainRules.bat	126
A.13	FR_COMPILE.bat	127
A.14	CCout.bat	127
A.15	dll.bat	127
A.16	COMPILE.bat	127
B. Publications by the Candidate Relevant to the Thesis		
B.1	Symposium & Conference Proceedings Publications	128
B.2	Journal Publications	128
B.3	Pending	129

LIST OF ABBREVIATIONS

A	amper, unit of the current
BP	Back Propagation
C	Charge and discharge current rate of a battery
dT/dt	delta temperature / delta time
dU/dt	delta voltage / delta time
FL	Fuzzy Logic
FNN	Fuzzy Neural Network
GA	Genetic Algorithm
I	Current
Li-ion	Lithium Ion
Li-ion polymer	Lithium Ion Polymer
NDV	Negative Delta Voltage
NFG	Neuro-Fuzzy-Genetic
NiCd	Nickel Cadmium
NiMH	Nickel Metal Hydride
NN	Neural Network
RFNN	Recurrent Fuzzy Neural Network
s	Second
SoC	State of Charge
T	Internal battery temperature
$T_{\text{end}} - T_{\text{start}}$	Temperature increase during battery charging
U	Battery voltage
V	Volt, unit of the voltage

LIST OF TABLES

Table 2.1	History of battery development	4
Table 2.2	Characteristics of commonly used rechargeable batteries	6
Table 2.3	Advantages and limitations of NiCd batteries	7
Table 2.4	Comparison of the methods for control	21
Table 3.1	Rules represented as a table	33
Table 3.2	Applied Fuzzy Rules	35
Table 6.1	The control rules	55
Table 6.2	Comparison of Intelligence NiCd chargers	69
Table 6.3	Comparison of the intelligent charger controllers mean square errors	71

LIST OF FIGURES

Figure 2.1	Cross-section of a classic NiCd cell	8
Figure 2.2	Typical Charge Characteristics	10
Figure 2.3	Typical Self-discharge Characteristics	10
Figure 2.4	Typical Discharge Characteristics (Comparison with Dry-cell)	11
Figure 2.5	Typical Cycle Life Characteristics	11
Figure 2.6	Typical Capacity Recovery After Storage	12
Figure 2.7	Temperature/Voltage vs SOC characteristics of a NiCd cell	17
Figure 2.8	Fuzzy Control of the process	20
Figure 2.9	ANFIS control of the process	20
Figure 2.10	Neuro-Fuzzy-Genetic control	21
Figure 2.11	NeuFuz system	22
Figure 2.12	The systems that been used by [3]	23
Figure 2.13	Fuzzy rules that been used in [7]	23
Figure 2.14	ANFIS model that been used by [9]	24
Figure 3.1	The architecture of the neuro fuzzy genetic battery charger	27
Figure 3.2	Temperature-to-Digital Converter	29
Figure 3.3	Triangle-shaped membership function	31
Figure 3.4	Trapezoidal membership function	31
Figure 3.5	Neural fuzzifier	32
Figure 3.6	Input Membership Functions: U , dU , T and dT	34
Figure 3.7	A Membership function of the output variable I	35
Figure 4.1	The structure of fully RFNN	40
Figure 4.2	GA based training of RFNN network	44
Figure 5.1	Voltage vs. Time & Temperature vs. Time Characteristic	46
Figure 5.2	Voltage model with Actual Voltage	49
Figure 5.3	Temperature model with Actual Temperature	50
Figure 5.4	Voltage characteristics for 8C, 5C and 3C	50
Figure 5.5	Temperature characteristics for 8C, 5C and 3C	51
Figure 6.1	Battery charging control process	59

Figure 6.2	Voltage and temperature	61
Figure 6.3	Derivative of voltage and temperature	62
Figure 6.4	Graph of the charging process	63
Figure 6.5	The control with modified temperature term membership functions	64
Figure 6.6	Intentionally remove some rules from consideration in control system	65
Figure 6.7	Intentionally remove some rules from consideration in control system	66
Figure 6.8	Corrupt membership functions of terms for the variables U and T	67
Figure 6.9	Results of a Charging process	68
Figure 6.10	Dynamic modeling of nonlinear systems using RFNN	70
Figure 6.11	The output of the plant ($y(k)$) and the output of RFNN ($\hat{y}(k)$)	71

1. INTRODUCTION

The nickel cadmium battery (commonly abbreviated NiCd or NiCad) is a popular type of rechargeable battery for portable electronics and toys using the metals nickel (Ni) and cadmium (Cd) as the active chemicals. NiCd batteries have a niche market in the area of cordless telephones, emergency lighting, as well as power tools. Due to their beneficial weight/energy ratio as compared to lead based technologies and good service lifetimes, NiCd batteries of large capacities with a wet electrolyte (wet NiCds) are used for electric cars and as start batteries for aero planes [31]. NiCd is the most popular battery type used by aerospace systems.

The NiCd battery charging is a nonlinear electrochemical dynamic process and for this reason it can be very difficult to predict in an accurate manner. The NiCd battery charging process has a high degree of uncertainty with no exact mathematical model. Traditionally, the models that have been used for electrochemical processes based on statistics, but these models do not approximate the dynamic behavior of the processes with the accuracy required in practice.

The research works on intelligent chargers have received considerable attention in recent literature. There are several intelligent research works on battery charging such as fuzzy, fuzzy-genetic and neuro-fuzzy. The existing systems minimize charging time or internal temperature increase and do not detect whether or not the battery is deeply discharged or shorted cell prior to charging. There is a danger that shorted cell batteries can blow up if forced to be charged. Thus, there is a need for an intelligent charger that minimizes both charging time & internal battery temperature increase, and detects the difference between deeply discharged and shorted cell.

The aim of the work that is presented in this thesis is to design and develop an intelligent NiCd battery charger that minimize both charging time & internal temperature increase and detect the difference between deeply discharged and shorted cell before starting to charge the battery. The novel proposed system uses neuro-fuzzy-

genetic approach for modeling and control NiCd battery charging process. The dynamics of NiCd battery are described using recurrent fuzzy neural network (RFNN) trained by genetic algorithm (GA) to generate a fuzzy rule base to control battery charging process. In addition, a dynamic data mining technique for extraction of control rules for effective and fast NiCd battery charging process is also suggested within this work. Receiving current fuzzy values of the input signals, the suggested control system performs fuzzy inference and determines fuzzy values of output control signal.

The proposed system is simulated using various software tools that include Java, C#, Visual Basic and Microsoft Excel. This thesis is organized as follows:

Chapter 2 outlines a detailed knowledge on batteries and charging, review of existing intelligent NiCd batteries chargers with discussion of their capabilities and limitations.

Chapter 3 presents a description of working principles, structure and elements of neuro-fuzzy-genetic control system for NiCd batteries charging.

Chapter 4 provides detailed description of the fuzzy neural networks and their learning process.

Chapter 5 describes the fuzzy modeling of the NiCd batteries charging process by using recurrent fuzzy neural network.

In Chapter 6, firstly the simulation results of the proposed approach are presented. Then a comparative analysis of neuro-fuzzy-genetic charger with existing intelligent ones is given. Then another comparison is presented between the proposed charger and the existing researches based on identification error.

Finally in the Conclusion, the results obtained in previous chapters are summarized, and ideas for the future research are given.

2. STATE OF THE ART INTELLIGENT BATTERY CHARGING SYSTEMS

2.1. Batteries as Control Object

A battery converts chemical energy into electric energy through an electrochemical process. The basic unit is called a "cell" and can be manufactured in a wide variety of shapes and sizes. Batteries are made up of one or more cells in series or parallel combinations to create the desired voltage and output capacity.

The electrochemical cells consists of two terminal suspended in an electrolyte. The terminals are called the anode and the cathode. An electrical current is essentially a flow of electrons, and the battery can be regarded as an electron pump. The chemical reaction between the anode and the electrolyte forces electrons out of the electrolyte and into the anode metal, through the circuit, then back to the cathode. From the cathode metal, the electrons re-enter the electrolyte. This direction may seem strange, from negative to positive. The current has been conventionally regarded as flowing from positive down to negative, but in fact; this current is a flow of electrons in the opposite direction. The anode and cathode both get converted during this reaction, one is 'eaten away', and the other has a build-up of material on it. When a rechargeable battery is recharged, this chemical reaction is reversed, and the terminals are restored.

Batteries can be divided into two classes: *primary*, and *secondary*. Primary batteries are designed for a single discharge cycle only, i.e. they are non-rechargeable. Secondary cells are designed to be recharged, typically, from 200 to 1000 times. The historical development of batteries is given in Table 2.1 [28]. The battery may be much ancient. It is believed that the Parthians who ruled Baghdad (250 BC) used batteries to electroplate silver. The Egyptians are said to have electroplated antimony onto copper over 4300 years ago.

Table 2.1 History of battery development [28].

1600	Gilbert (UK)	Establishment electrochemistry study
1791	Galvani (Italy)	Discovery of 'animal electricity'
1800	Volta (Italy)	Invention of the voltaic cell
1802	Cruikshank (UK)	First electric battery capable of mass production
1820	Ampère (France)	Electricity through magnetism
1833	Faraday (UK)	Announcement of Faraday's Law
1836	Daniell (UK)	Invention of the Daniell cell
1859	Planté (France)	Invention of the lead acid battery
1868	Leclanché (France)	Invention of the Leclanché cell
1888	Gassner (USA)	Completion of the dry cell
1899	Jungner (Sweden)	Invention of the nickel-cadmium battery
1901	Edison (USA)	Invention of the nickel-iron battery
1932	Shlecht & Ackermann (Germany)	Invention of the sintered pole plate
1947	Neumann (France)	Successfully sealing the NiCd battery
Mid 1960	Union Carbide (USA)	Development of primary alkaline battery
Mid 1970		Development of valve regulated lead acid battery
1990		Commercialization nickel-metal hydride battery
1992	Kordesch (Canada)	Commercialization reusable alkaline battery
1999		Commercialization lithium-ion polymer
2001		Anticipated volume production of proton exchange membrane fuel cell

2.1.1 Battery Chemistries

Advanced battery systems offer very high energy densities, deliver 1000 charge/discharge cycles and are paper thin. Batteries are scrutinized not only in terms of energy density but service life, load characteristics, maintenance requirements, self-discharge and operational costs. Since NiCd remains a standard against which other

batteries are compared. Let us evaluate alternative chemistries against this classic battery type.

- Nickel Cadmium (NiCd) — mature and well understood but relatively low in energy density. The NiCd is used where long life, high discharge rate and economical price are important. Main applications are two-way radios, biomedical equipment, professional video cameras and power tools. The NiCd contains toxic metals and is not environmentally friendly.
- Nickel-Metal Hydride (NiMH) — has a higher energy density compared to the NiCd at the expense of reduced cycle life. NiMH contains no toxic metals. Applications include mobile phones and laptop computers.
- Lead Acid — most economical for larger power applications where weight is of little concern. The lead acid battery is the preferred choice for hospital equipment, wheelchairs, emergency lighting and UPS systems.
- Lithium Ion (Li-ion) — fastest growing battery system. Li-ion is used where high-energy density and light weight is of prime importance. The Li-ion is more expensive than other systems and must follow strict guidelines to assure safety. Applications include notebook computers and cellular phones.
- Lithium Ion Polymer (Li-ion polymer) — a potentially lower cost version of the Li-ion. This chemistry is similar to the Li-ion in terms of energy density. It enables very slim geometry and allows simplified packaging. Main applications are mobile phones.
- Reusable Alkaline — replaces disposable household batteries; suitable for low-power applications. Its limited cycle life is compensated by low self-discharge, making this battery ideal for portable entertainment devices and flashlights.

The characteristics of these six most commonly used rechargeable battery systems compared at Table 2.2 given in terms of energy density, cycle life, exercise requirements and cost [28]. The table is based on average ratings of commercially available batteries.

Table 2.2 Characteristics of commonly used rechargeable batteries [28].

	NiCd	NiMH	Lead Acid	Li-ion	Li-ion polymer	Reusable Alkaline
Gravimetric Energy Density (Wh/kg)	45-80	60-120	30-50	110-160	100-130	80 (initial)
Internal Resistance (includes peripherals circuits) mW	100 to 200 6V pack	200 to 300 6V pack	<100 12V pack	150 to 250 7.2V pack	200 to 300 7.2V pack	200 to 2000 6V pack
Cycle Life (to 80% of initial capacity)	1500	300 to 500	200 to 300	500 to 1000	300 to 500	50 (to 50%)
Fast Charge Time	1h typical	2-4h	8-16h	2-4h	2-4h	2-3h
Overcharge Tolerance	moderate	low	high	very low	low	moderate
Self-discharge / Month (at room temperature)	20%	30%	5%	10%	~10%	0.3%
Cell Voltage (nominal)	1.25V	1.25V	2V	3.6V	3.6V	1.5V
Load Current - peak - best result	20C 1C	5C 0.5C or lower	5C 0.2C	>2C 1C or lower	>2C 1C or lower	0.5C 0.2C or lower
Operating Temperature (discharge only)	-40 to 60°C	-20 to 60°C	-20 to 60°C	-20 to 60°C	0 to 60°C	0 to 65°C
Maintenance Requirement	30 to 60 days	60 to 90 days	3 to 6 months	not req.	not req.	not req.
Typical Battery Cost (US\$, reference only)	\$50 (7.2V)	\$60 (7.2V)	\$25 (6V)	\$100 (7.2V)	\$100 (7.2V)	\$5 (9V)
Cost per Cycle (US\$)	\$0.04	\$0.12	\$0.10	\$0.14	\$0.29	\$0.10-0.50
Commercial use since	1950	1990	1970	1991	1999	1992

Note that NiCd has the shortest charge time, delivers the highest load current and offers the lowest overall cost-per-cycle, but has the most demanding maintenance requirements.

2.1.2 The Nickel Cadmium Battery

Alkaline nickel battery technology is originated in 1899, when Waldmar Jungner invented the NiCd battery. The materials were expensive compared to other battery types available at the time and its use was limited to special applications. In 1932, the active materials were deposited inside a porous nickel-plated electrode and in 1947, research began on a sealed NiCd battery, which recombined the internal gases generated during charge rather than venting them. These advances led to the modern sealed NiCd battery. The advantages and limitations of NiCd Batteries are given in Table 2.3 [28].

Table 2.3 Advantages and limitations of NiCd batteries

Advantages	<ul style="list-style-type: none"> • Fast and simple charge — even after prolonged storage.
	<ul style="list-style-type: none"> • High number of charge/discharge cycles — if properly maintained, the NiCd provides over 1000 charge/discharge cycles.
	<ul style="list-style-type: none"> • Good load performance — the NiCd allows recharging at low temperatures.
	<ul style="list-style-type: none"> • Long shelf life – in any state-of-charge.
	<ul style="list-style-type: none"> • Simple storage and transportation — most airfreight companies accept the NiCd without special conditions.
	<ul style="list-style-type: none"> • Good low temperature performance.
	<ul style="list-style-type: none"> • Forgiving if abused — the NiCd is one of the most rugged rechargeable batteries.
	<ul style="list-style-type: none"> • Economically priced
	<ul style="list-style-type: none"> • Available in a wide range of sizes and performance options — most NiCd cells are cylindrical.
Limitations	<ul style="list-style-type: none"> • Relatively low energy density — compared with newer systems.
	<ul style="list-style-type: none"> • Memory effect
	<ul style="list-style-type: none"> • Environmentally unfriendly — the NiCd contains toxic metals. Some countries are limiting the use of the NiCd battery.
	<ul style="list-style-type: none"> • Has relatively high self-discharge — needs recharging after storage.

The NiCd prefers fast charge to slow charge and pulse charge to DC charge. All other chemistries prefer a shallow discharge and moderate load currents. The NiCd is the only battery type that performs best under rigorous working conditions. A periodic full discharge is so important that, if omitted, large crystals will form on the cell plates (also referred to as 'memory') and the NiCd will gradually lose its performance.

Among rechargeable batteries, NiCd remains a popular choice for applications such as two-way radios, emergency medical equipment, professional video cameras and power tools. Most of the rechargeable batteries for portable equipment are NiCd. However, the introduction of batteries with higher energy densities and less toxic metals is causing a diversion from NiCd to newer technologies.

At the beginning battery cells were encased in glass jars. Later, larger batteries were developed that used wooden containers. The inside was treated with a sealant to prevent electrolyte leakage. With the need for portability, the cylindrical cell appeared. After World War II, these cells became the standard format for smaller, rechargeable batteries. Figure 2.1 [4] illustrates the conventional cell of a NiCd battery.

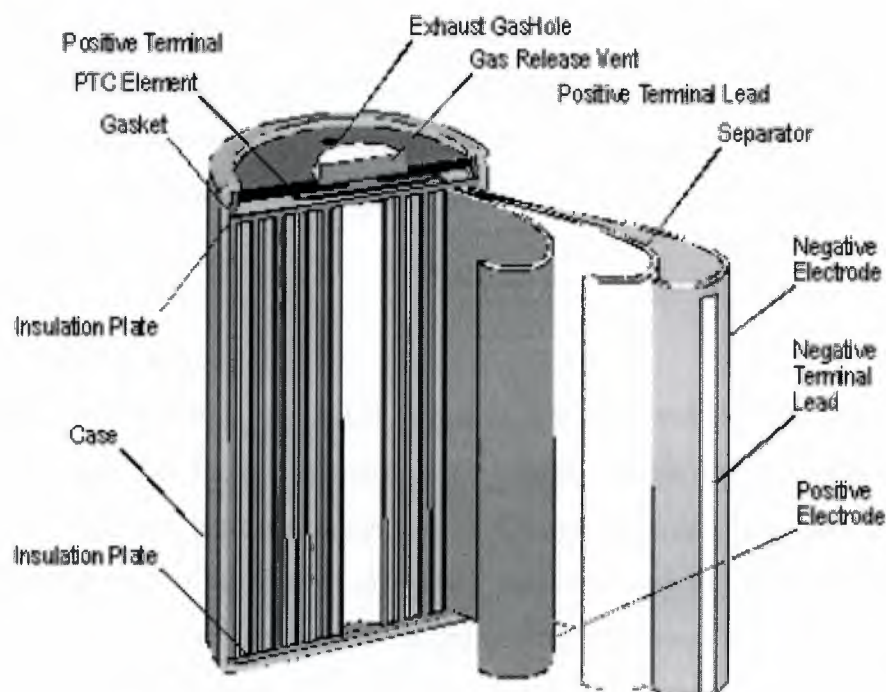


Figure 2.1 Cross-section of a classic NiCd cell [4]

The cylindrical cell is moderately priced and offers high energy density. Typical applications are wireless communication, mobile computing, biomedical instruments, power tools and other uses that do not demand ultra-small size. NiCd offers the largest selection of cylindrical cells.

Nickel-based cells provide a nominal cell voltage of 1,25V. Nickel-based cells are often marked 1,2V. There is no difference between a 1,2 and 1,25V cell; it is simply the preference of the manufacturer in marking. Whereas commercial batteries tend to be identified with 1,2V/cell, industrial, aviation and military batteries are still marked with the original designation of 1,25V/cell. A five-cell nickel-based battery delivers 6V (6,25V with 1,25V/cell marking) and a six-cell pack has 7,2V (7,5V with 1,25V/cell marking). Packs with fewer cells in series generally perform better than those with 12 cells or more.

On higher voltage batteries, precise cell matching becomes important, especially if high load currents are drawn or if the pack is operated in cold temperatures. Parallel connections are used to obtain higher ampere-hour (Ah) ratings. When possible, pack designers prefer using larger cells. This may not always be practical because new battery chemistries come in limited sizes. Often, a parallel connection is the only option to increase the battery rating. Paralleling is also necessary if pack dimensions restrict the use of larger cells. Among the battery chemistries, Li-ion lends itself best to parallel connection. NiCd batteries have five main characteristics: charge, discharge, cycle life, storage, and safety.

a) Charge Characteristics

The charge characteristics of NiCd batteries are affected by the current, time, temperature, and other factors. Increasing the charge current and lowering the charge temperature causes the battery voltage to rise. Charge generates heat, thus causing the battery temperature to rise. Charge efficiency will also vary according to the current, time, and temperature. For rapid charge, a charge control system is required; refer to the following section on the charge methods for NiCd batteries. A typical charge characteristic for a Panasonic NiCd battery is shown in Figure 2.2 [5].

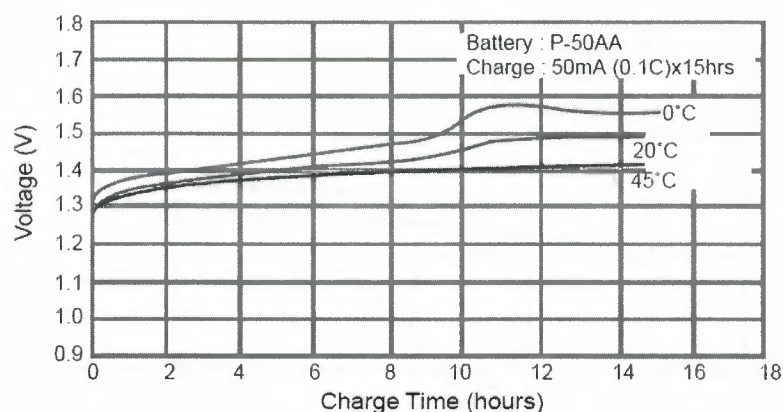


Figure 2.2 Typical Charge Characteristics

b) Discharge Characteristics

The discharge characteristics of NiCd batteries will vary according to the current, temperature, and other factors. Generally, in comparison with dry-cell batteries, there is less voltage fluctuation during discharge, and even if the discharge current is high, there is very little drop in capacity. Among the various types of NiCd batteries, there are models such as Panasonic's "P" type which are specifically designed to meet the need for high-current discharge, such as for power tools, and there are also models such as new High Capacity and Rapid Charge type which are designed to meet the need for high capacity, such as for high-tech devices. A typical self discharge characteristics is shown in Figure 2.3 and the comparison with dry-cell from [5] shown in Figure 2.4.

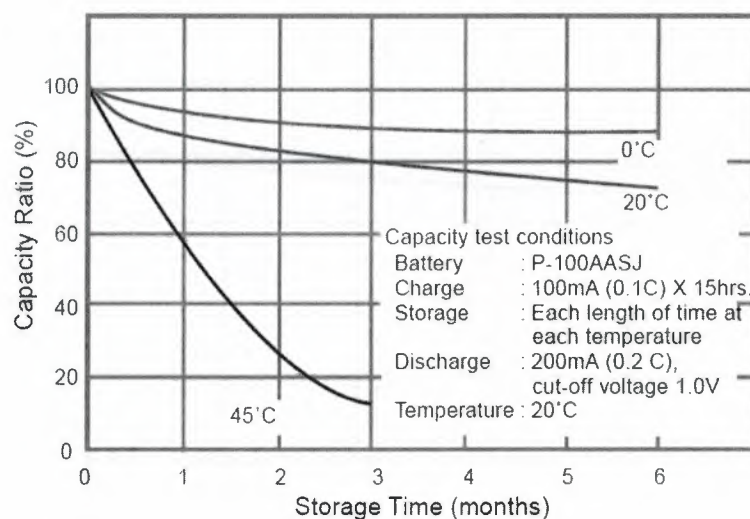


Figure 2.3 Typical Self-discharge Characteristics

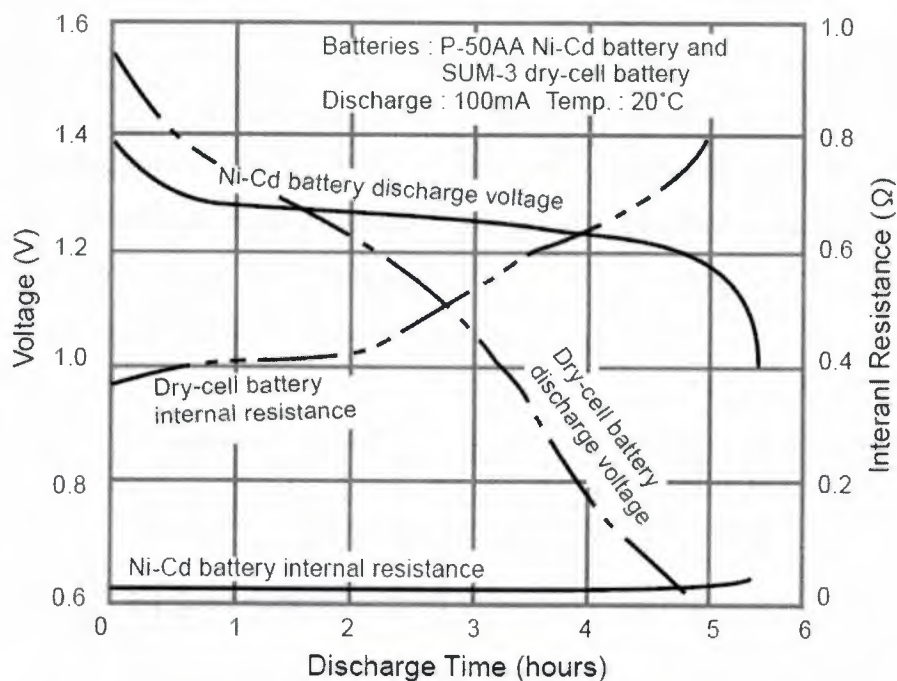


Figure 2.4 Typical Discharge Characteristics (Comparison with Dry-cell)

c) Cycle Life Characteristics

The cycle life of NiCd batteries will vary according to the charge and discharge conditions, the temperature, and other usage conditions. The actual cycle life will vary according to which of the various charge formats is used, such as for rapid charge, and also according to how the device powered by the batteries is actually used. A typical cycle life characteristics from [5] shown in Figure 2.5.

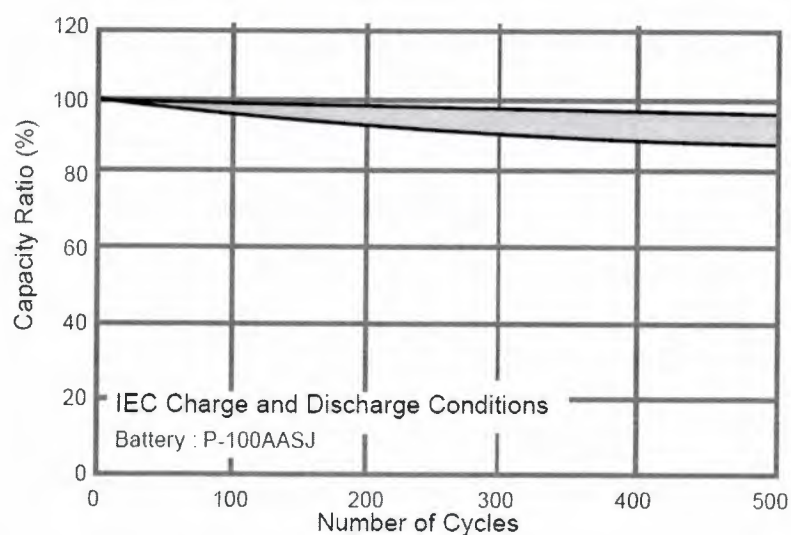


Figure 2.5 Typical Cycle Life Characteristics

d) Storage Characteristics

When NiCd batteries are stored in a charged state, the capacity will gradually decrease (self discharge), and this tendency will be markedly greater at high temperatures. However, the capacity can be subsequently restored by charge. Even if the batteries are stored for an extended length of time, if the storage conditions are appropriate, the capacity will be restored by subsequent charge and discharge. A typical capacity recovery after storage from [5] shown in Figure 2.6.

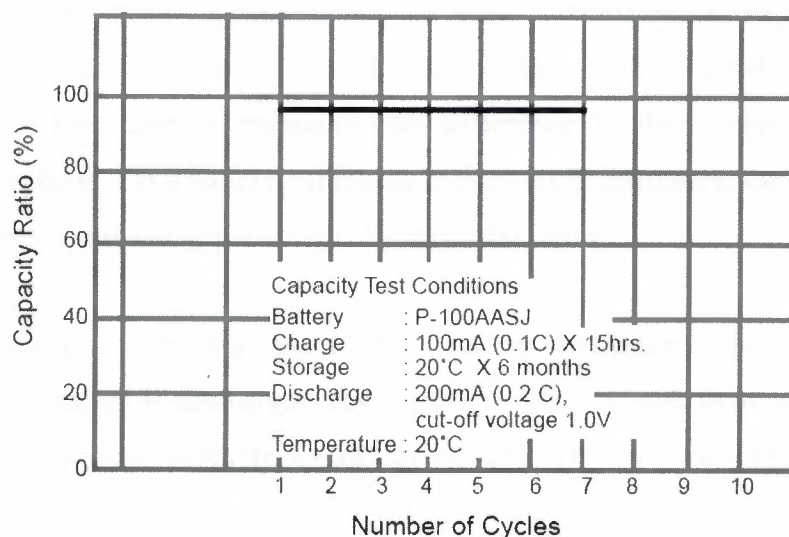


Figure 2.6 Typical Capacity Recoveries after Storage

e) Safety

If pressure inside the battery rises as a result of improper use, such as overcharge, short-circuit, or reverse charge, a reset able safety valve will function to release the pressure, thus preventing bursting of the battery.

2.1.3 Charging the Nickel Cadmium Battery

The charge and discharge current of a battery is measured in C-rate. Most portable batteries, with the exception of the lead acid, are rated at 1C. A discharge of 1C draws a current equal to the rated capacity. For example, a battery rated at 1000mAh provides 1000mA for one hour if discharged at 1C rate. The same battery discharged at 0.5C provides 500mA for two hours. At 2C, the same battery delivers 2000mA for

30 minutes. The capacity of a battery is commonly measured with a battery analyzer. If the analyzer's capacity readout is displayed in percentage of the nominal rating, 100 percent is shown if 1000mA can be drawn for one hour from a battery that is rated at 1000mAh. If the battery only lasts for 30 minutes before cutoff, 50 percent is indicated. A new battery sometimes provides more than 100 percent capacity. In such a case, the battery is conservatively rated and can endure a longer discharge time than specified by the manufacturer.

The discrepancy in capacity readings with different C-rates largely depends on the internal resistance of the battery. On a new battery with a good load current characteristic or low internal resistance, the difference in the readings is only a few percentage points. On a battery exhibiting high internal resistance, the difference in capacity readings could swing plus/minus 10 percent or more.

Applying the capacity offset does not improve battery performance; it merely adjusts the capacity calculation if discharged at a higher or lower C-rate than specified. The battery manufacturer determines the amount of capacity offset recommended for a given battery type.

Battery manufacturers recommend that new batteries be slow charged for 24 hours before use. A slow charge helps to bring the cells within a battery pack to an equal charge level because each cell self-discharges to different capacity levels. During long storage, the electrolyte tends to gravitate to the bottom of the cell. The initial trickle charge helps redistribute the electrolyte to remedy dry spots on the separator that may have developed.

Some battery manufacturers do not fully form their batteries before shipment. These batteries reach their full potential only after the customer has primed them through several charge/discharge cycles, either with a battery analyzer or through normal use. In many cases, 50 to 100 discharge/charge cycles are needed to fully form a nickel-based battery. Quality cells, such as those made by Sanyo [4], Panasonic [5] and Energizer [6], are known to perform to full specification after as few as 5 to 7 discharge/charge

cycles. Early readings may be inconsistent, but the capacity levels become very steady once fully primed. A slight capacity peak is observed between 100 and 300 cycles.

Most rechargeable cells are equipped with a safety vent to release excess pressure if incorrectly charged. The safety vent on a NiCd cell opens at 1034 to 1379 kPa (150 to 200 psi). In comparison, the pressure of a car tire is typically 240 kPa (35 psi). With a releasable vent, no damage occurs on venting but some electrolyte is lost and the seal may leak afterwards. When this happens, a white powder will accumulate over time at the vent opening.

Commercial fast-chargers are often not designed in the best interests of the battery. This is especially true of NiCd chargers that measure the battery's charge state solely through temperature sensing. Although simple and inexpensive in design, charge termination by temperature sensing is not accurate. The thermistors used commonly exhibit broad tolerances; their positioning with respect to the cells are not consistent. Ambient temperatures and exposure to the sun while charging also affect the accuracy of full-charge detection. To prevent the risk of premature cut-off and assure full charge under most conditions, charger manufacturers use 50°C as the recommended temperature cut-off. Although a prolonged temperature above 45°C is harmful to the battery, a brief temperature peak above that level is often unavoidable.

More advanced NiCd chargers sense the rate of temperature increase, defined as dT/dt , or the change in temperature over charge time, rather than responding to an absolute temperature (dT/dt is defined as $\Delta \text{Temperature} / \Delta \text{time}$). This type of charger is kinder to the batteries than a fixed temperature cut-off, but the cells still need to generate heat to trigger detection. To terminate the charge, a temperature increase of 1°C per minute with an absolute temperature cut-off of 60°C works well. Because of the relatively large mass of a cell and the sluggish propagation of heat, the delta temperature, as this method is called, will also enter a brief overcharge condition before the full-charge is detected. The dT/dt method only works with fast chargers.

Harmful overcharge occurs if a fully charged battery is repeatedly inserted for topping charge. Vehicular or base station chargers that require the removal of two-way radios

with each use are especially hard on the batteries because each reconnection initiates a fast-charge cycle. This also applies to laptops that are momentarily disconnected and reconnected to perform a service. Repetitive connection to power affects mostly 'dumb' nickel-based batteries. A 'dumb' battery contains no electronic circuitry to communicate with the charger.

More precise full charge detection of nickel-based batteries can be achieved with the use of a micro controller that monitors the battery voltage and terminates the charge when a certain voltage signature occurs. A drop in voltage signifies that the battery has reached full charge. This is known as Negative Delta V (NDV). NDV is the recommended full-charge detection method for 'open-lead' NiCd chargers because it offers a quick response time. The NDV charge detection also works well with a partially or fully charged battery. If a fully charged battery is inserted, the terminal voltage raises quickly, then drops sharply, triggering the ready state. Such a charge lasts only a few minutes and the cells remain cool. NiCd chargers based on the NDV full charge detection typically respond to a voltage drop of 10 to 30mV per cell. Chargers that respond to a very small voltage decrease are preferred over those that require a larger drop.

To obtain a sufficient voltage drop, the charge rate must be 0.5C and higher. Lower than 0.5C charge rates produce a very shallow voltage decrease that is often difficult to measure, especially if the cells are slightly mismatched. In a battery pack that has mismatched cells, each cell reaches the full charge at a different time and the curve gets distorted. Failing to achieve a sufficient negative slope allows the fast-charge to continue, causing excessive heat buildup due to overcharge. Chargers using the NDV must include other charge-termination methods to provide safe charging under all conditions. Most chargers also observe the battery temperature.

The charge efficiency factor of a standard NiCd is better on fast charge than slow charge. At a 1C charge rate, the typical charge efficiency is 1.1 or 91 percent. On an overnight slow charge (0.1C), the efficiency drops to 1.4 or 71 percent. At a rate of 1C, the charge time of a NiCd is slightly longer than 60 minutes (66 minutes at an assumed charge efficiency of 1.1). The charge time on a battery that is partially discharged or

cannot hold full capacity due to memory or other degradation is shorter accordingly. At a 0.1C charge rate, the charge time of an empty NiCd is about 14 hours, which relates to the charge efficiency of 1.4.

During the first 70 percent of the charge cycle, the charge efficiency of a NiCd battery is close to 100 percent. Almost all of the energy is absorbed and the battery remains cool. Currents of several times the C-rating can be applied to a NiCd battery designed for fast charging without causing heat build-up. Ultra-fast chargers use this unique phenomenon and charge a battery to the 70 percent charge level within a few minutes. The charge continues at a lower rate until the battery is fully charged. Once the 70 percent charge threshold is passed, the battery gradually loses ability to accept charge. The cells start to generate gases, the pressure rises and the temperature increases. The charge acceptance drops further as the battery reaches 80 and 90 percent SoC. Once full charge is reached, the battery goes into overcharge. In an attempt to gain a few extra capacity points, some chargers allow a measured amount of overcharge. Figure 2.7 illustrates the relationship of cell voltage, pressure and temperature while a NiCd is being charged [5].

Ultra-high capacity NiCd batteries tend to heat up more than the standard NiCd if charged at 1C and higher. This is partly due to the higher internal resistance of the ultra-high capacity battery. Optimum charge performance can be achieved by applying higher current at the initial charge stage, then tapering it to a lower rate as the charge acceptance decreases. This avoids excess temperature rise and yet assures fully charged batteries.

The cell voltage, pressure and temperature characteristics are similar in a NiMH cell. Interspersing discharge pulses, between charge pulses improves the charge acceptance of nickel-based batteries. Commonly referred to as 'burp' or 'reverse load' charge, this charge method promotes high surface area on the electrodes, resulting in enhanced performance and increased service life. Reverse load also improves fast charging because it helps to recombine the gases generated during charge. The result is a cooler and more effective charge than with conventional DC chargers.

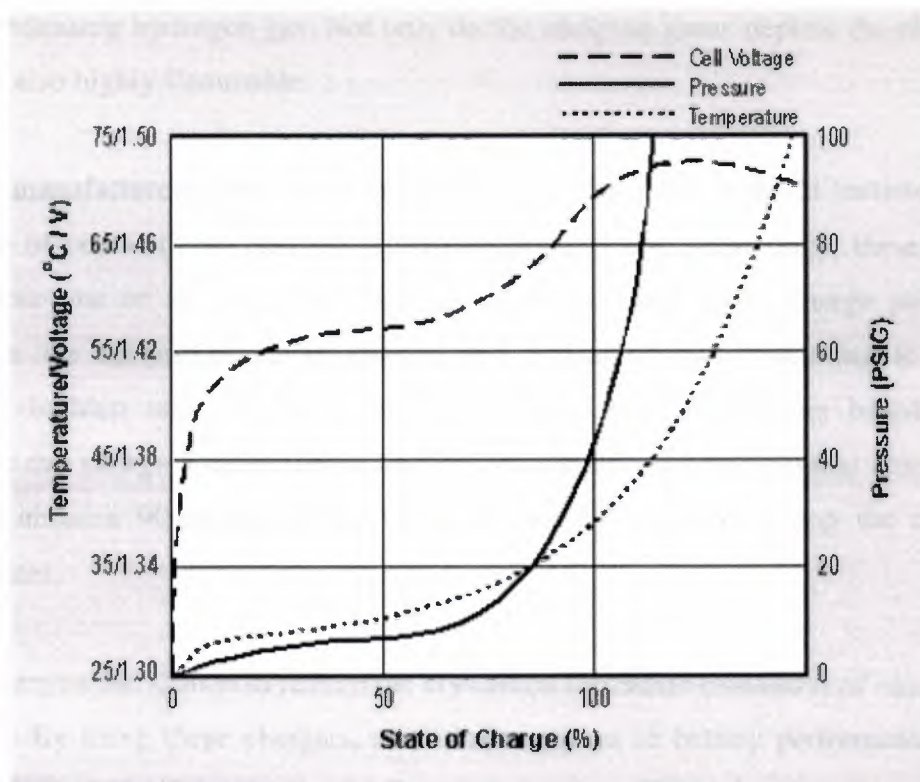


Figure 2.7 Temperature/Voltage v SOC characteristics of a NiCd cell.

After full charge, the NiCd battery is maintained with a trickle charge to compensate for the self-discharge. The trickle charge for a NiCd battery ranges between 0.05C and 0.1C. In an effort to reduce the memory phenomenon, there is a trend towards lower trickle charge currents.

Some charger manufacturers claim amazingly short charge times of 30 minutes or less. With well-balanced cells and operating at moderate room temperatures, NiCd batteries designed for fast charging can indeed be charged in a very short time. This is done by simply dumping in a high charge current during the first 70 percent of the charge cycle. Some NiCd batteries can take as much a 10C, or ten times the rated current. Precise SoC detection and temperature monitoring are essential.

The high charge current must be reduced to lower levels in the second phase of the charge cycle because the efficiency to absorb charge is progressively reduced as the battery moves to a higher SoC. If the charge current remains too high in the later part of the charge cycle, the excess energy turns into heat and pressure. Eventually venting

occurs, releasing hydrogen gas. Not only do the escaping gases deplete the electrolyte, they are also highly flammable.

Several manufacturers offer chargers that claim to fully charge NiCd batteries in half the time of conventional chargers. Based on pulse charge technology, these chargers intersperse one or several brief discharge pulses between each charge pulse. This promotes the recombination of oxygen and hydrogen gases, resulting in reduced pressure buildup and a lower cell temperature. Ultra-fast-chargers based on this principle can charge a nickel-based battery in a shorter time than regular chargers, but only to about a 90 percent SoC. A trickle charge is needed to top the charge to 100 percent.

Pulse chargers are known to reduce the crystalline formation (memory) of nickel-based batteries. By using these chargers, some improvement in battery performance can be realized, especially if the battery is affected by memory. The pulse charge method does not replace a periodic full discharge. For more severe crystalline formation on nickel-based batteries, a full discharge or recondition cycle is recommended to restore the battery.

Ultra-fast charging can only be applied to healthy batteries and those designed for fast charging. Some cells are simply not built to carry high current and the conductive path heats up. The battery contacts also take a beating if the current handling of the spring-loaded plunger contacts is underrated. Pressing against a flat metal surface, these contacts may work well at first, and then wear out prematurely. Often, a fine and almost invisible crater appears on the tip of the contact, which causes a high resistive path or forms an isolator. The heat generated by a bad contact can melt the plastic.

Another problem with ultra-fast charging is servicing aged batteries that commonly have high internal resistance. Poor conductivity turns into heat, which further deteriorates the cells. Battery packs with mismatched cells pose another challenge. The weak cells holding less capacity are charged before those with higher capacity and start to heat up. This process makes them vulnerable to further damage.

Many of today's fast chargers are designed for the ideal battery. Charging less than perfect specimens can create such a heat buildup that the plastic housing starts to distort. Provisions must be made to accept special needs batteries, albeit at lower charging speeds. Temperature sensing is a prerequisite.

The ideal ultra-fast charger first checks the battery type, measures its SoH and then applies a tolerable charge current. Ultra-high capacity batteries and those that have aged are identified, and the charge time is prolonged because of higher internal resistance. Such a charger would provide due respect to those batteries that still perform satisfactorily but are no longer 'spring chickens'.

The charger must prevent excessive temperature build-up. Sluggish heat detection, especially when charging takes place at a very rapid pace, makes it easy to overcharge a battery before the charge is terminated. This is especially true for chargers that control fast charge using temperature sensing alone. If the temperature rise is measured right on the skin of the cell, reasonably accurate SoC detection is possible. If done on the outside surface of the battery pack, further delays occur. Any prolonged exposure to a temperature of 45°C harms the battery.

New charger concepts are being studied which regulate the charge current according to the battery's charge acceptance. On the initial charge of an empty battery when the charge acceptance is high and little gas is generated, a very high charge current can be applied. Towards the end of a charge, the current is tapered down.

2.2 Review of Existing Work of Intelligent Batteries Chargers

The dynamics of an electrochemical system is non-linear and the mathematical models are difficult to derive. There are several intelligence works on battery charging process.

Castillo & Melin implemented three different intelligent systems by MATLAB in [1] to control a complex electrochemical process. The authors of [1] compared the results of fuzzy (Figure 2.8), neuro-fuzzy (Figure 2.9) and neuro-fuzzy-genetic systems (Figure

2.10) with conventional PID control by simulating the formation (loading) of a battery. These systems designed using absolute temperature (T) and temperature gradient (dT/dt) as inputs and current (I) as output. The authors of [1] used a simple linear repression model as:

$$T = 88.03 + 2.5304 I \quad (2.1)$$

where 88.03 and 2.5304 are estimated parameters to be using real data.

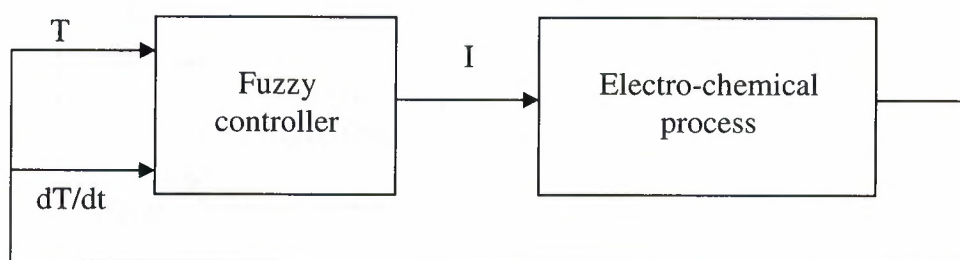


Figure 2.8 Fuzzy Control of the Process [1]

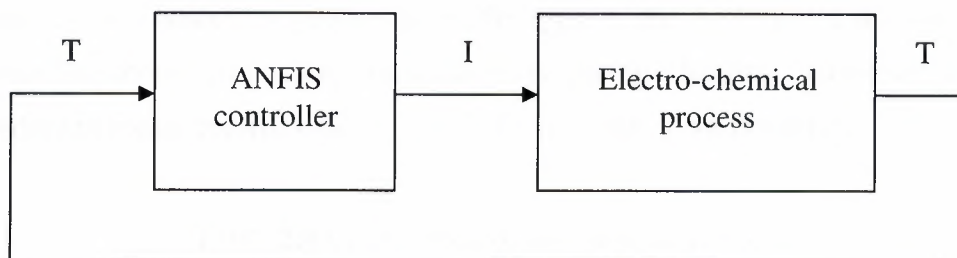


Figure 2.9 ANFIS Control of the Process

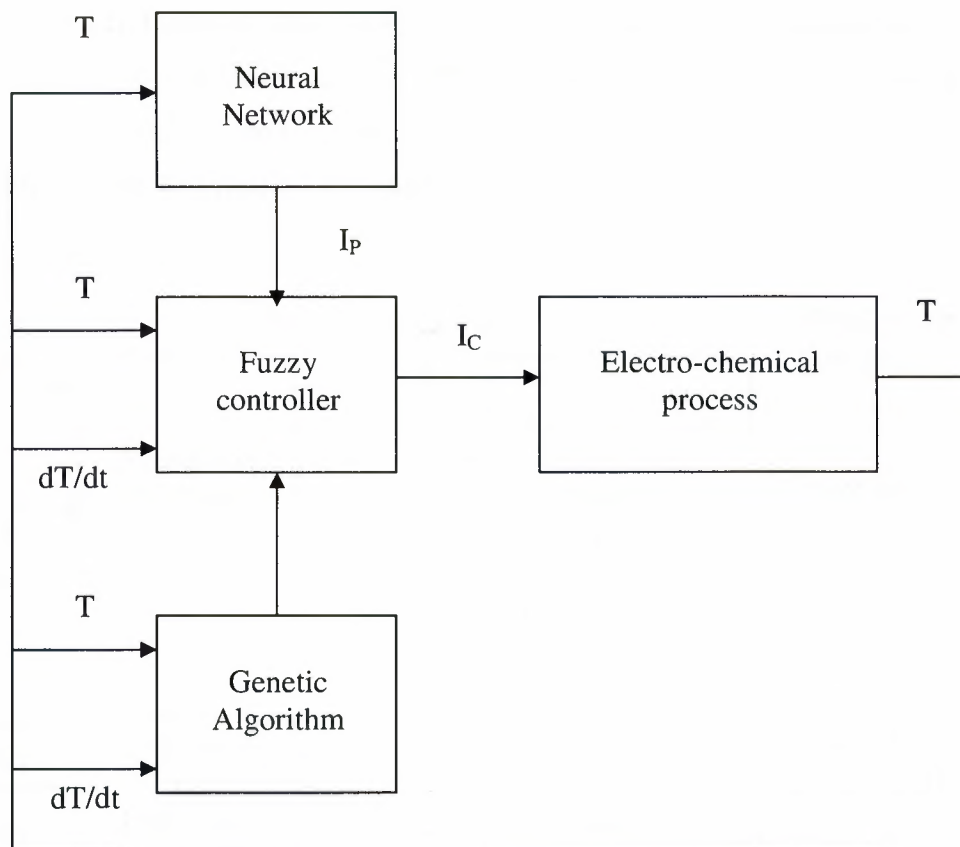


Figure 2.10 Neuro-Fuzzy-Genetic Control

According to Table 2.4 [1], neuro-fuzzy-genetic approach gives the best result to produce a battery in the manufacturing plant. Although [1] explains the duration of charging, unfortunately it neither gives the type of the battery nor does it give any information about temperature increase level during charging. However, a specific procedure to create control system for battery charging is not presented.

Table 2.4 Comparison of the methods for control

Control Method	Time for Loading (hours)
Manual Control	50
Conventional Control	36
Fuzzy Control	32
Neuro-Fuzzy Control	30
Neuro-Fuzzy-Genetic	25

the experiments instead of designing a model as shown in Figure 2.12. This paper gives low level $T_{\text{end}} - T_{\text{start}}$ and a short charging time results unfortunately not the lowest ones.

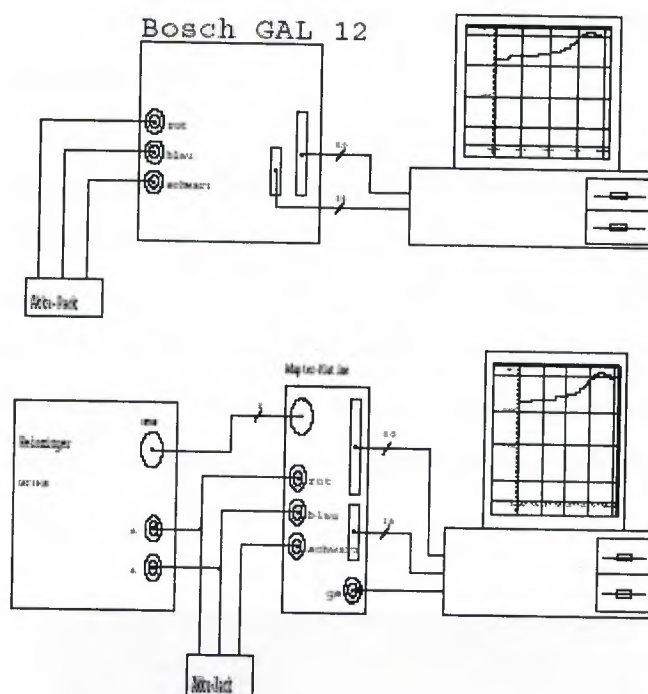


Figure 2.12 The systems that been used by [3]

Authors of [7], propose some characteristics and implement the means on rule editor of the MATLAB instead of designing a specific NiCd battery model. The inputs of the designed fuzzy system are T , dT/dt , U and dU/dt where the output is I . In the conclusion the authors of [7] gives the $T_{\text{end}} - T_{\text{start}}$ result as 35 up to 60 degree Celsius. The charging time is not mentioned by the authors in [7]. Although the authors suggested that their system increases the life time up to 3000 cycle, they do not give the initial life time. The $T_{\text{end}} - T_{\text{start}}$ result is also too high compared with other research papers.

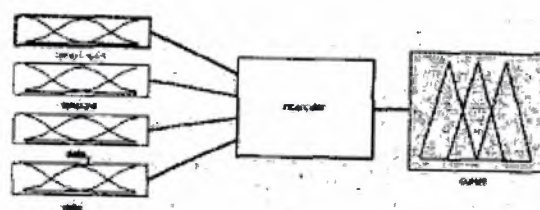


Figure 2.13 Fuzzy rules that been used in [7]

The Author of [8] pointed out that the 'intelligent' battery chargers will not be able to detect the difference between deeply discharged and shorted cell batteries. In this paper the proposed method to solve this problem is checking the conductance. Unfortunately conductance controlled battery charging needs to measure the capacitance of the battery that increases the charging time.

Paper [9] considers a fuzzy controller for rapid NiCd batteries charger using adaptive Neuro-Fuzzy inference system (ANFIS). The NiCd batteries were charged at different rates between 8 and 0.05 C-rate and for different durations. The two input variables identified to control the C are T and dT/dt . The equivalent ANFIS architecture for the system under consideration is shown in Figure 2.14 using MATLAB. Although this work gives the best result on charging time which gives a high level 50 degree Celsius $T_{end} - T_{start}$.

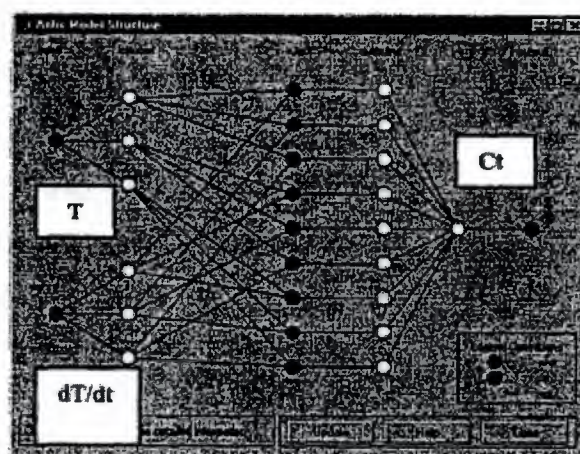


Figure 2.14 ANFIS model that been used by [9]

Authors of [10], control the battery charging process by a microcontroller. T and U are used to control the output I. However the total charging time and $T_{end} - T_{start}$ results are not presented in this work.

As it is mentioned in the [11] there is no comparison between recent research works on intelligent chargers of NiCd batteries.

2.3. Statement of Present Work Problem

The aim of this dissertation is to design a controller that gives both least charging time and least temperature increase together and also able to detect the difference between deeply discharged and shorted cells. The system will also provide means to avoid charging shorted batteries.. To reach these goals the dissertation focuses on designing a model for nickel cadmium battery charging process and then uses this model to design a controller.

The purpose of battery control system is to charge the whole battery pack, consisting of 8 battery cells to hold 9,6V. The initial charge level is 1,37V and temperature is 21,6°C. Just after the battery reaches 1,6V, it becomes overheated and loss of charge is observed due to some chemical processes inside the battery. The purpose of control system is to charge the battery to hold 1,6V in a possibly shorter time while preventing the battery from overheating. Different charging I values can be applied to battery from controller output.

The input signals T and U are defining the crisp current values of the battery temperature and voltage respectively. The battery charging control system measures temperature and voltage sensors. To consider the battery dynamics better, the first derivatives of U (dU/dt) and T (dT/dt) are used as additional input signals to the battery charging controller.

As it is mentioned above, there is no available formal mathematical model of the battery under the charging process. A Soft Computing based computational model is required to allow dynamic update through the life of the battery. To design the required charger a RFNN is used to learn the required behavior of the battery charging system. The fuzzy rule extraction is performed to generate a set of fuzzy rules and membership functions. The acquired knowledge is then analyzed and adjusted by human and incorporated into fuzzy logic based controller.

All the input signals: U, dU/dt , T and dT/dt , are fuzzified to compute relevance to respective fuzzy terms used in the rules. The controller performs fuzzy inference and

determines fuzzy values of control signal. The defuzzified fuzzy control signal, representing the value of current (I), from fuzzy logic based charging controller is then applied to the battery.

3. ARCHITECTURE OF NEURO-FUZZY-GENETIC CONTROL SYSTEM FOR BATTERIES CHARGING

3.1 Structure of Control System and Description of its Working Principles

The input signals of suggested control system for batteries charging temperature (T) and voltage (U) are measured by temperature and voltage sensors. Outputs of the sensors are crisp current values of temperature and voltage. As one can see in Figure 3.1 other input signals of neuro-fuzzy-genetic controller for battery charging is first derivatives of U (dU/dt) and first derivatives of T (dT/dt). All these input signals U , dU/dt , T and dT/dt are transmitted into fuzzy signals by fuzzifiers. Knowledge base of neuro-fuzzy-genetic controller is implemented by RFNN approximately. Receiving current fuzzy values of U , dU/dt , T and dT/dt controller performs fuzzy inference and determines fuzzy values of control signal. As only crisp control signals are applied to battery obtained from RFNN fuzzy control signal must be defuzzified by defuzzifier. This signal is transmitted from analog to digital and applied to the battery [18-27].

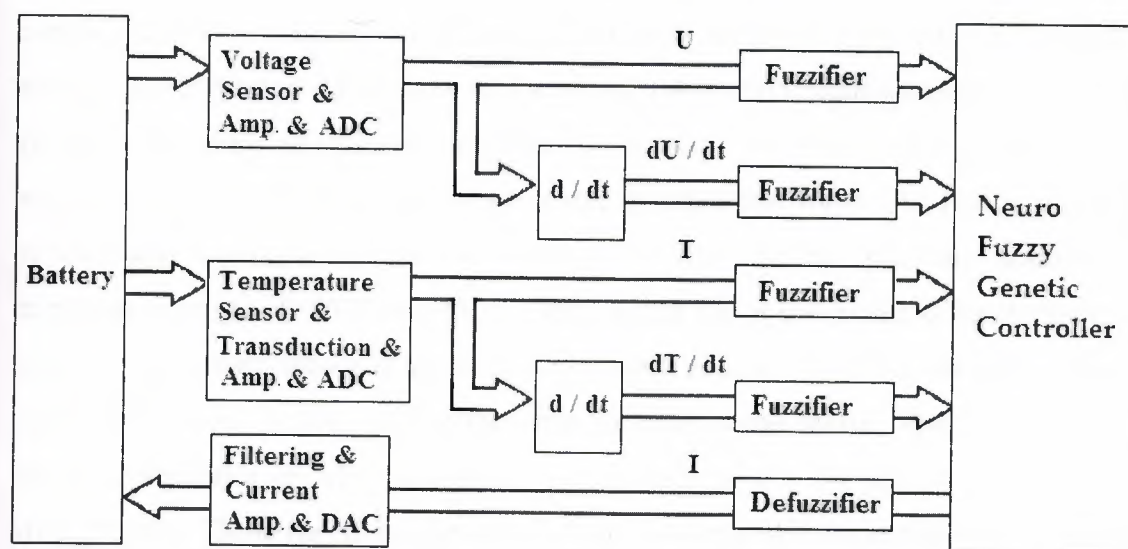


Figure 3.1 The architecture of the neuro fuzzy genetic battery charger

3.2 Elements of Batteries Charging Control Systems

In this project, during simulation of the controller it was assumed that the temperature and voltage values measured by sensors are converted to digital signals before entering the fuzzifiers. The output current filtered, amplified and then converted to analog signal after defuzzifier.

An analog-to-digital converter (abbreviated ADC, A/D or A to D) is an electronic circuit that converts continuous signals to discrete digital numbers. The reverse operation is performed by a digital-to-analog converter. A digital-to-analog converter (DAC or D-to-A) is a device for converting a digital (usually binary) code to an analog signal (current in Figure 3.1). Digital-to-Analog Converters are the interface between the abstract digital world and the analog real life, See [31] for more details.

Several temperature sensing techniques are currently in widespread usage as explained. The most common of these are Resistance Temperature Detectors (RTDs), thermocouples, thermistors, and sensor ICs. Resistive sensors use a sensing element whose resistance varies with temperature. A platinum RTD consists of a coil of platinum wire wound around a bobbin, or a film of platinum deposited on a substrate. Another type of resistive sensor is the thermistor. Low-cost thermistors often perform simple measurement or trip-point detection functions in low-cost systems. A thermistors resistance-temperature function is very nonlinear. A thermocouple consists of a junction of two wires made of different materials. Integrated circuit temperature sensors differ significantly from the other types in a couple of important ways. The first is operating temperature range. A temperature sensor IC can operate over the nominal IC temperature range of -55°C to $+150^{\circ}\text{C}$. The second major difference is functionality. A silicon temperature sensor is an integrated circuit, and can therefore include extensive signal processing circuitry within the same package as the sensor. There is no need to design comparator or ADC circuits to convert their analog outputs to logic levels or digital codes. Those functions are already built into several commercial ICs. For details see [29].

The circuit has shown in Figure 3.2 [29] implements a low-cost system for measuring temperature at several points within a system and converts the temperature readings to digital form. With the components shown here, up to 19 LM45 temperature sensors drive separate inputs of an ADC08019 8-bit, 19-channel ADC with serial (microwire, SPI) data interface. The tiny SOT-23 sensor packages allow the designer to place the sensors in virtually any location within the system. The 1,28V reference voltage is chosen to provide a conversion scale of $1 \text{ LSB} = 5\text{mV} = 0,5^{\circ}\text{C}$, with full-scale equal to 128°C . The R-C network at the sensor output provides protection against oscillation if capacitive loads (or cables) may be encountered, and also help filter output noise. The reference voltage can be manually adjusted to 1,28V with the 10k potentiometer, or the potentiometer can be replaced with a fixed resistor. If 5% values will be used, a 3,3 k Ω resistor will work. For better accuracy, use 1% resistors; the pot can then be replaced by a 3,24 k Ω resistor.

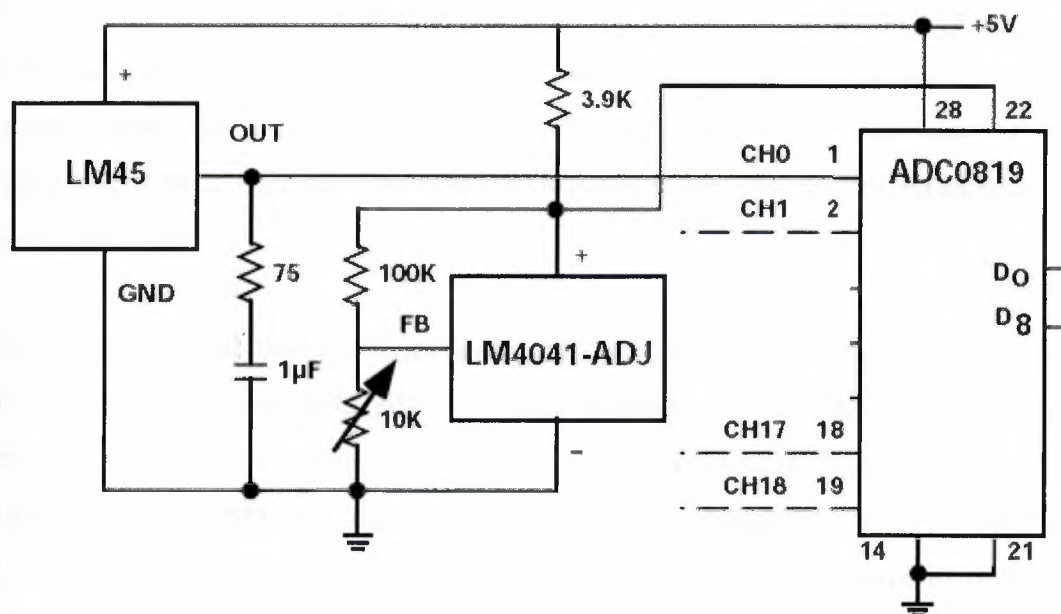


Figure 3.2 Analog-to-Digital Converters

A voltage sensor is a sensor used for measuring voltage. They are used for measuring DC or AC voltage. Voltage sensors, also known as electrical voltage sensors, are available in either digital or analog type. Digital- type voltage sensors are more accurate than analog- type voltage sensors. Voltage Sensors can be mounted either in printed circuit boards (PCB) or DIN rails. Voltage sensors can be used in combination with

current sensors. Voltage Sensors are available in different voltage range. The input of the voltage sensor may be DC input or AC input. Voltage sensors can work in the absence of input. Differential voltage sensors are used to measure voltage. The differential voltage sensor consists of a sensor that has a small signal amplifier with a wide range of frequency. The sensor is used for measuring voltage in DC and AC circuits. The sensor takes differential voltage as input and produces negative and positive potentials. Differential voltage sensors are used in parallel with circuit elements. Potential difference between the ends is measured by the differential voltage sensor. The measured voltage is then passed to the amplifier unit and adjusted to the required range. The differential voltage sensor also comes with over-voltage protection for safety purposes. Interfaces are available for differential voltage sensors, see [30] for more details.

The difference of the fuzzification and defuzzification used in neuro-fuzzy systems from those used traditionally is in that here the role of adjusting or learning is more important. A fuzzifier is a unit that has one input and several outputs; each of the outputs represents a certain fuzzy term and returns the membership value of a given input to that term. As a rule, fuzzifiers are adjustable and can be realized as neurons and neural networks.

By analogy, a unit that allows obtaining an appropriate crisp value based on a given fuzzy one is named defuzzifier. Several methods of defuzzification are known; difference among them is in the way they calculate the average value based on the area below the membership function curve. There are no recommendations on which formula is the most universal and precise - each separate application may have its own reasons for selecting one out of these three. An attractive feature of the defuzzification in neuro-fuzzy systems is that the formula of defuzzification can be adjusted just like the membership function.

The method to create fuzzifier (and also defuzzifier) depends mostly on the requirements for the shapes of generated membership functions, that is on the formula that allows general appearance for membership function shapes necessary for this

application. On the base of the obtained formula one can select the necessary type of neuron.

For triangle-shaped membership function that been used for current (I) output membership function , the appropriate fuzzifier is the neuron implementing the function

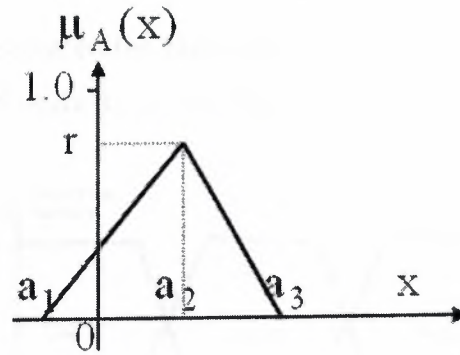


Figure 3.3 Triangle-shaped membership function

$$Y = \max(\min(\frac{x - a_2}{a_2 - a_1}, \frac{x - a_3}{a_2 - a_3}), 0) \quad (3.1)$$

This neuron has three parameters, directly defining the associated fuzzy number by the triple (a_1, a_2, a_3) , i.e. left bound, middle point and right bound.

For trapezoidal membership functions that used for T , dT/dt , U and dU/dt membership functions , the appropriate fuzzifier is the neuron implementing the function

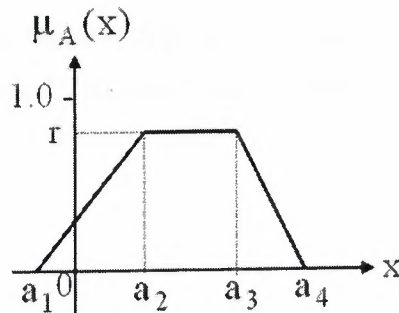


Figure 3.4 Trapezoidal membership function

$$\mu_A(x) = \begin{cases} \frac{x-a_1}{a_2-a_1}r, & \text{if } a_1 \leq x \leq a_2, \\ r, & \text{if } a_2 \leq x \leq a_3, \\ \frac{a_4-x}{a_4-a_3}r, & \text{if } a_3 \leq x \leq a_4, \\ 0, & \text{otherwise} \end{cases} \quad (3.2)$$

The machine, having replaced the fuzzifier by a traditional neural network and learned it, will obtain so-called neural fuzzifier (Figure 3.5).

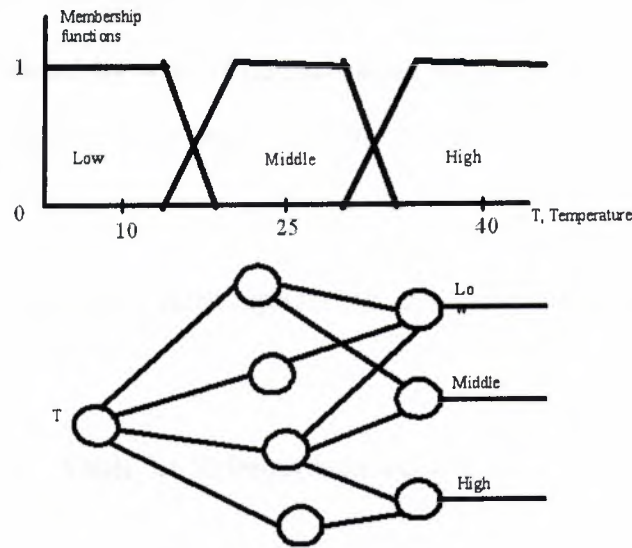


Figure 3.5. Neural fuzzifier

The most commonly used defuzzification formula that consists in finding the line that divides the area below the membership function (describing the defuzzified number) into two equal-size sub-areas. The associated method is known by name "Center-of-gravity method".

$$y_{\text{defuz}} = \frac{\int y \cdot \mu(y) dy}{\int \mu(y) dy}. \quad (3.3)$$

Because the output membership functions may consist of a combination (union) of some parts of input membership functions, and this combination does not in general inherit the features of the original shapes then the resulting defuzzifying function will

hardly be of a compact mathematical form. So the appropriate defuzzifier is built to substituted a unit performing the function.

However, in most cases, the output membership functions are expressed as singletons especially when concerned with the monolith (neuro-and fuzzy in a single chip) compact, simple and effective neuro-fuzzy systems for limited applications. Then the defuzzifier is built.

In this case the output membership functions are assumed to be singletons. They are completely defined by the parameters of the neuron (W_j) after learning. Here the division may be excluded by way of normalization of the coefficients (W_j), i.e. the defuzzification is performed by a neuron that just weighs the input signal forming its output.

In the latter case the generated rules represented as a table (multi-dimensional in the general case) like Table 3.1.

Table 3.1 Rules represented as a table

Temperature	Low	Average	High
Low	0.19	0.25	0.15
Average	0.05	0.31	0.12
High	0.28	0.01	0.09

As the block diagram shows in Figure 3.1, the U, dU, T and dT applied to the neuro-fuzzy-genetic charger and the output current (I) connected to the charger to charge the battery. Different charging currents values have been applied ranging from 0A to 6A that allows flexibility in tailoring the charge level to the state of the battery to achieve an optimal charging scheme [2]. The first few simulations give unsatisfactory results. In the next simulation cycles, fuzzy rules and membership functions were changed to get the best results from the system. The input membership functions U, dU, T and dT

used in simulation are given in Figure 3.6 and the output membership function I is given in Figure 3.7.

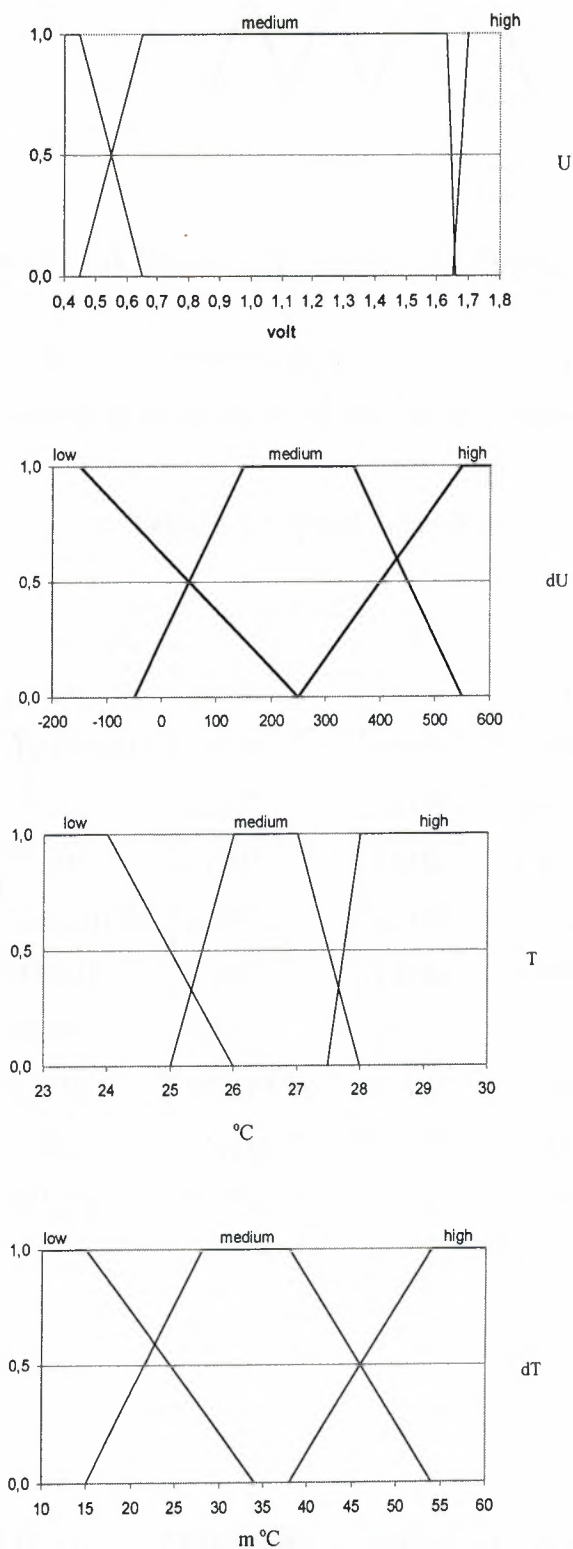


Figure 3.6 Input Membership Functions: U , dU , T and dT .

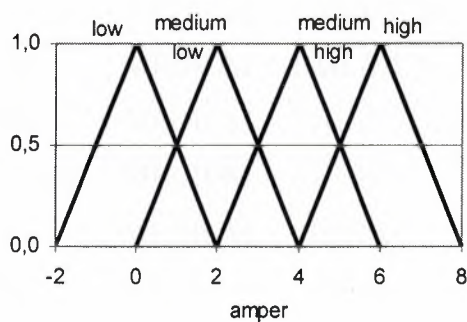


Figure 3.7 A Membership function of the output variable I

The knowledgebase designed according to the NiCd charge characteristics as explained in section 2.1.3. The rules given in Table 3.2. are applied to the system at the beginning.

Table 3.2 Applied Fuzzy Rules

T	dT/dt	U	dU/dt	I
LOW	LOW	LOW	LOW	HIGH
LOW	MEDIUM	LOW	LOW	HIGH
LOW	HIGH	LOW	LOW	HIGH
MEDIUM	LOW	LOW	LOW	HIGH
MEDIUM	MEDIUM	LOW	LOW	MEDIUM HIGH
MEDIUM	HIGH	LOW	LOW	MEDIUM LOW
HIGH	ANY	ANY	ANY	LOW
LOW	LOW	MEDIUM	LOW	HIGH
LOW	MEDIUM	MEDIUM	LOW	HIGH
LOW	HIGH	MEDIUM	LOW	HIGH
.				
LOW	HIGH	MEDIUM	MEDIUM	HIGH
MEDIUM	LOW	MEDIUM	MEDIUM	MEDIUM HIGH

MEDIUM	MEDIUM	MEDIUM	MEDIUM	MEDIUM HIGH
MEDIUM	HIGH	MEDIUM	MEDIUM	MEDIUM LOW
LOW	LOW	MEDIUM	HIGH	HIGH
LOW	MEDIUM	MEDIUM	HIGH	HIGH
LOW	HIGH	MEDIUM	HIGH	HIGH
MEDIUM	LOW	MEDIUM	HIGH	MEDIUM HIGH
MEDIUM	MEDIUM	MEDIUM	HIGH	MEDIUM HIGH
MEDIUM	HIGH	MEDIUM	HIGH	MEDIUM LOW

4. RECURRENT FUZZY NEURAL NETWORKS AND THEIR LEARNING

4.1 Review on Fuzzy Neural Networks

Fuzzy logic has been successfully applied in many areas, such as control of industrial processes, control of robotic manipulators, control of servo-motors, complex decision making, diagnostic systems and others. When using fuzzy logic, input data in the form of linguistic values are represented by membership functions which are used for defining fuzzy sets of crisp values and their corresponding membership degrees related to these sets. Many parameters of systems based on fuzzy logic should be defined with help of an expert. In the same time, however, the parameters associated in the construction of processes are performed by a method of the trial and error or some heuristic algorithms. Moreover, a designer who knows the characteristics of the system also needs to specify initial rules. Some researchers have suggested a number of mechanisms to generate fuzzy rules and developed methods of their modification on the basis of their experience. Among them, we must distinguish the self-organizing fuzzy controllers that are capable of forming and modifying fuzzy rules, the clustering algorithms for fuzzy partitioning of an input data space, and the least square algorithm for defining a succession of parameters that can be used to construct systems based on fuzzy logic.

Despite utilization of all these algorithms, they still remain to be heuristic in some sense, and the selection of membership functions in fuzzy rules is connected to the method of trial and error. In order to remove this disadvantage recently some interesting methods were developed that utilize the learning ability of neural networks. Multi-layer neural networks with such learning methods as backpropagation, reinforcement, and unsupervised algorithms, or the combinations of the above, have been used to select fuzzy rules and precisely adjust membership functions used in those rules. For fuzzy neural network, the signals and/or the weights must be fuzzy set. There are three types of fuzzy neural networks (FNN)- FNN_1 , FNN_2 , FNN_3 .

The first type of fuzzy neural net (abbreviated FNN_1) has crisp signals (X_i) ($i = \overline{1, n}$) but fuzzy weights. The second type of FNN (FNN_2) has fuzzy signals and real number weights. The last FNN (FNN_3) has both fuzzy signals and fuzzy weights.

4.2 Description of Investigated Recurrent Fuzzy Neural Networks

In spite of great importance of fuzzy feed-forward and recurrent neural networks (FNN) to solve a wide range of real world problems, today there is no effective learning algorithm for FNN.

In [38] a learning algorithm for FNN is proposed with fuzzy signals and weights called the "fuzzified delta rule". As shown in [39] this algorithm lacks theoretical basis and is derived only through the use of the real differentiation instead of the fuzzy differentiation.

In [40] an architecture and learning algorithm of FNN are proposed with trapezoid fuzzy weights. Using Zadeh's extension principle, α -level sets and interval arithmetic, a crisp learning algorithm is derived for adjusting four parameters of each trapezoid fuzzy weight. Earlier in [41] similar crisp learning algorithm for FNN was proposed with triangular weights. In [39, 42] the idea of using genetic algorithms was expressed for learning of FNN.

In this section we consider an effective genetic-based learning mechanism for FNN with fuzzy inputs, fuzzy weights expressed as LR-fuzzy numbers, and fuzzy outputs. Hamming fuzzy distance, which is non-differentiable is used as a performance error index for learning of FNN. The main distinguishing features of the proposed learning method from the existing approaches include the following:

1. The proposed method does not require differentiability of error performance index of FNN.
2. Fuzzy is used distance in general fuzzy number space for comparison of fuzzy actual fuzzy output and target of FNN. Therefore the derived algorithm is not crisp. The

proposed method guarantees a higher degree than existing algorithms in the global minimum of error performance of FNN.

4.3 Genetic Algorithm Based Learning of Recurrent Fuzzy Neural Networks

Fuzzy neural network (FNN) approach becomes a powerful tool for solving real-world problems in area of forecasting, identification, control, image recognition and others that are related with high level uncertainty [13]. This is related with the fact that the FNN combines the capacity of fuzzy reasoning in handling uncertain information and the capability of pure neural networks in the learning from experiments. The advantage of FNN is that it allows automation of design of fuzzy rules and combined learning of numerical data as well as expert knowledge expressed as fuzzy IF-THEN rules. FNN may have smaller network size and be faster in convergence speed as compared with ordinary NN.

In spite of great importance of fuzzy feed-forward and recurrent neural networks there is no effective learning algorithms for FNN. There are two approaches for training FNN. First approach is level-sets of fuzzy numbers and application of the back-propagation (BP) learning algorithm. The second approach is to involve Genetic Algorithm (GA) to minimize error function and determine the fuzzy connection weights and biases. The main advantage of the GA based approach is that it is less complicate yet guarantees with higher degree the global minimum of total error performance index than classic gradient-based error minimization methods applied to alpha cuts [13]. In case of dynamic or temporal problems there is a need for RFNN.

Because of the advantages of FNN and GA based approach that mentioned above we decide to design our NiCd battery charger with GA based trained RFNN.

The structure of the considered RFNN is presented in Figure 4.1. The box elements represent memory cells that store values of activation of neurons at previous time step, which is feed back to the input at the next time step.

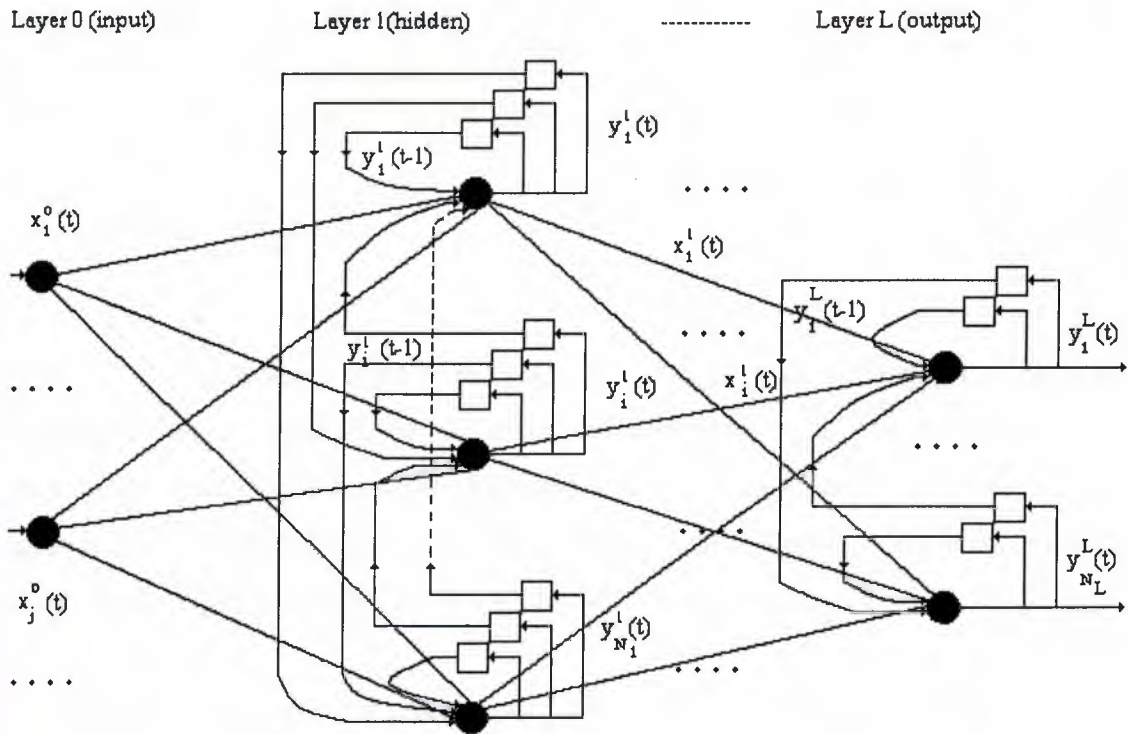


Figure 4.1 The structure of fully RFNN

In general the network may have virtually any number of layers successively from 0 (the first or input layer) to L (last or output layer). The neurons in the first (layer 0) layer are only distributing the input signals without modifying the values

$$y_i^0(t) = x_i^0(t) \quad (4.43)$$

The neurons in the next layer 1 have dynamic links and process the incoming fuzzy signals from layer 0 by the following formula:

$$y_i^1(t) = F(\theta_i^1) + \sum_j x_j^1(t) w_{ij}^1 + \sum_j y_j^1(t-1) v_{ij}^1 \quad (4.44)$$

The neurons in the remaining hidden layers (layer 2 to layer N_L-2) are also dynamic and compute their output signals as follows

$$y_i^l(t) = F(\theta_i^l) + \sum_j x_j^l(t)w_{ij}^l + \sum_j y_j^l(t-1)v_{ij}^l \quad (4.45)$$

The output neurons (layer L) are non-dynamic linear neurons:

$$y_i^L(t) = F(\theta_i^L) + \sum_j x_j^L(t)w_{ij}^L \quad (4.46)$$

where $x_j^l(t)$ is j-th fuzzy input to the neuron I at layer l at the time step t, $y_j^l(t)$ is the computed output signal of the neuron at the step t, w_{ij} is the fuzzy weight of the connection to neuron I from neuron j located at the previous layer, θ_i is the fuzzy bias of neuron i, $y_j^l(t-1)$ is the activation of neuron j at the time step (t-1), v_{ij} is the recurrent connection weight to neuron i from neuron j at the same layer.

The activation function F for a total input to the neuron s is calculated as:

$$F(s) = \frac{s}{1 + |s|} \quad (4.47)$$

All fuzzy signals and connection weights and biases are general fuzzy numbers that with any required precision can be represented as $T(L_0, L_1, \dots, L_{n-1}, R_{n-1}, R_{n-2}, \dots, R_0)$. In case the original learning patterns are crisp, we need to sample data into fuzzy terms, i.e. to fuzzify the learning patterns. The fuzzifiers can be created independently for specific problems.

To apply genetic algorithm based approach for RFNN training, all adjustable parameters i.e. connection weights and biases are coded as bitstrings. A combination of all weight and bias bitstrings compose a genome (sometimes called a chromosome) representing a potential solution to the problem.

During the genetic evolution a population consisting a set of individuals or genomes (usually 50-100 genomes) undergo a group of operations with some genetic operators, most used of which are crossover and mutation. Applying genetic operators results in generating many offsprings (new individuals or genomes).

When bitstrings are decoded back to weights and biases, presenting different RFNN solutions, some may present good network solutions and some bad. Good genomes (i.e. those corresponding to good solutions) have more changes to keep within the populations for many generations ahead while bad genomes have more changes to be discarded during the selection process.

Whether a genome is good or bad is evaluated by a fitness function. The fitness function is an evaluator function (can also be fuzzy) numerically evaluating the quality of genome and the representing solution. In case of a NN learning this paper's purpose is to minimize the network error performance index.

Therefore the selection of best genomes from the population is done on the basis of the genome fitness value, which is calculated from RFNN error performance index. The calculation of fitness value of a particular genome require restoration of the coded genome bits back to fuzzy weight coefficients and biases of RFNN in other words it need to get a phenotype from the genotype.

The RFNN error performance index can be calculated as follows:

$$E_{tot} = \sum_p \sum_i D(y_{pi}, y_{pi}^{des}), \quad (4.48)$$

$$D(T1, T2) = \sum_{i=0}^{i=n-1} k_i |L_{T1i} - L_{T2i}| + \sum_{i=0}^{i=n-1} k_i |R_{T1i} - R_{T2i}|,$$

where E_{tot} is the total error performance index for all output neurons i and all learning data entries p ; $D(T1, T2)$ is the distance measure between two fuzzy numbers $T1$ and $T2$; $0 \leq k_0 \leq k_1 \dots \leq k_{n-2} \leq k_{n-1}$ are some scaling coefficients. Once the total error performance index for a combination of weights has been calculated the fitness f of the corresponding genome is set as:

$$f = \frac{1}{1 + E_{tot}} \quad (4.49)$$

As can be seen, the fitness function value for a genome (coding a network solution) is based on a distance measure comparing two sets of fuzzy values. Scaling coefficients are included to add sensitivity to high membership areas of a fuzzy number.

The GA used here can be described as follows:

1. Prepare the genome structure according to the structure of RFNN;
2. In case of existence of a good genome (an existing network solution), put it into population; else generate a random network solution and put it into population;
3. Generate at random new $PopSize-1$ genomes and put them into population;
4. Apply genetic crossover operation to $PopSize$ genomes in the population;
5. Apply mutation operation to the generated offsprings;
6. Get phenotype and rank (i.e. evaluate and assign fitness values to) all the offsprings;
7. Create new population with N_{best} best parent genomes and $(PopSize - N_{best})$ best offsprings;
8. Display fitness value of the best genome;
 - If termination condition is met go to Step 9;
 - Else go to step 4;
9. Get phenotype of the best genome in the population. Store network weights file;
10. Stop.

In above algorithm $PopSize$ is minimum population size and N_{best} is the number of best parent genomes always kept in the newly generated population.

The learning may be stopped once we see the process does not show any significant change in fitness value during many succeeding regenerations. In this case we can specify new mutation (and maybe crossover) probability and continue the process. If the obtained total error performance index or the behavior of the obtained network is not desired, we can restructure the network by adding new hidden neurons, or do better sampling (fuzzification) of the learning patterns. The GA-based training process is schematically shown in Figure 4.2.

5. FUZZY MODELING OF THE NICD BATTERIES BY USING RFNN

5.1 Approximation of the Nonlinear Dynamics of the NiCd Battery by RFNN

The battery converts chemical energy into electrical energy through an electro-chemical process. The basic unit is called a "cell" and can be manufactured in a wide variety of shapes and sizes. Batteries are made up of one or more cells in series or parallel combinations to create the desired voltage and output capacity. The dynamics of the electrochemical systems are nonlinear and therefore their mathematical models are difficult to derive. There are several intelligent methods for battery modeling explained in section 2.2.

RFNN based model of NiCd batteries is created on base of larger experimental analysis. These experiments included voltage-temperature characteristics for different C-rates. Figure 5.1 shows the voltage vs. time and temperature vs. time curves of a NiCd battery charging process when applied 6C (the initial charge level is 1,37V and temperature is 21,6°C). Just after the battery reaches 1,6V, it becomes overheated and we can observe loss of charge due to some chemical processes inside the battery.

At the beginning of the charging, the temperature does not rise significantly but remains relatively flat. After the battery reaches 80% of its capacity, its temperature begins to rise sharply. At the end of charging the voltage falls because of the temperature as shown in Figure 5.1. An internal temperature increase forces the internal resistance to decrease. The purpose of control is to charge the battery to hold 1,6V in a possibly shorter time while preventing the battery from overheating.

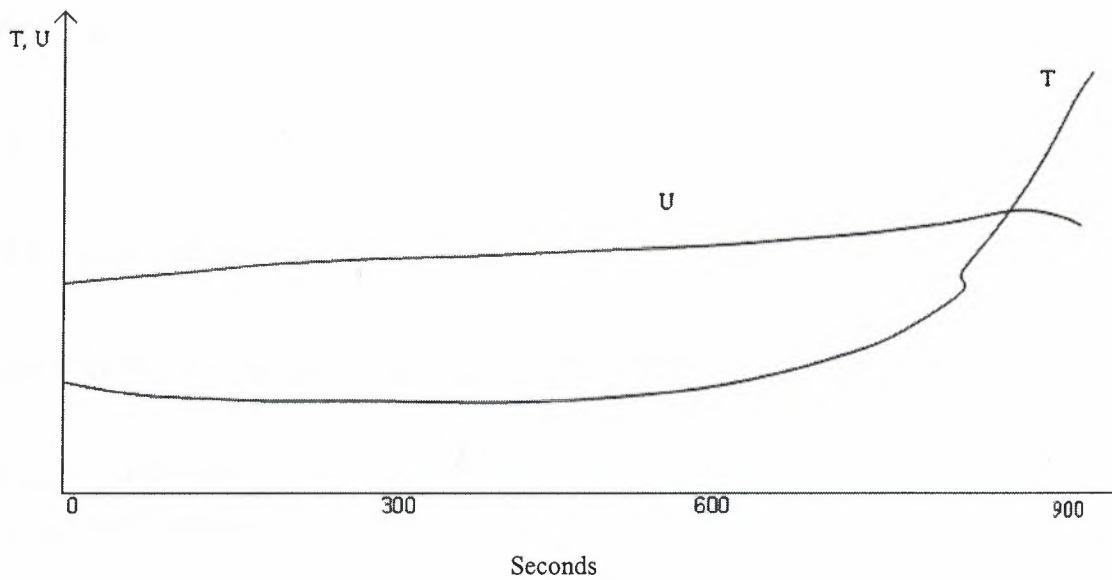


Figure 5.1 Voltage v time & temperature v time characteristic [3].

During the charging process, beside the main electrochemical reactions, oxidation and corrosion occur especially during overcharging. These undesired reactions make the mathematical analysis very difficult [3]. There is no exact mathematical model. As a result, instead of designing a nonlinear differential equation model, it is preferable to use neuro-fuzzy-genetic method to define a model.

To design the charger, NN is used to learn the behavior of the battery charging and to generate a set of fuzzy rules and membership functions, then acquired knowledge into new fuzzy logic system. The system creates the following voltage and temperature models.

- Voltage Model

$U = R_U(NN_U(S_U(\tau)))$ where

$$NN_U(X) = F(\theta_1^{(2)} + w_1^{(2)} F(\theta_1^{(1)} + w_1^{(1)} x) + w_2^{(2)} F(\theta_2^{(1)} + w_2^{(1)} x) + w_3^{(2)} F(\theta_3^{(1)} + w_3^{(1)} x))$$

the fitness function is

$$F(y) = y / (1 + |y|)$$

$$\text{with } S_U(\tau) = -0.75 + 1.5(\tau - \tau_{\min}) / (\tau_{\max} - \tau_{\min})$$

$$\text{that } \tau_{\max} = 1050 \text{ and } \tau_{\min} = 0$$

$$\text{also } R_U(u) = U_{\min} + 2/3 (U_{\max} - U_{\min}) (0.75 + u) \text{ where}$$

$$U_{\max} = 1.593333333$$

$$U_{\min} = 1.362666667$$

finally the weights and biases are

$$w_1^{(1)} = -8.128643533155916$$

$$w_1^{(2)} = -2.4197691196295974$$

$$w_2^{(1)} = 8.697642485563088$$

$$w_2^{(2)} = 7.418189748923732$$

$$w_3^{(1)} = 9.392557862982816$$

$$w_3^{(2)} = -7.849240230740181$$

$$\theta_1^{(1)} = -5.662001088311867$$

$$\theta_1^{(2)} = -2.7049977000721745$$

$$\theta_2^{(1)} = -5.176190575873721$$

$$\theta_3^{(1)} = -6.166325547059995$$

- Temperature model

$$T = R_T(NN_T(S_T(\tau))) \text{ where}$$

$$NN_U(X) = F(\theta_1^{(2)} + w_1^{(2)} F(\theta_1^{(1)} + w_1^{(1)} x) + w_2^{(2)} F(\theta_2^{(1)} + w_2^{(1)} x) + w_3^{(2)} F(\theta_3^{(1)} + w_3^{(1)} x))$$

the fitness function is

$$F(y) = y / 1 + |y|$$

$$\text{with } S_T(\tau) = -0,75 + 1,5(\tau - \tau_{\min}) / (\tau_{\max} - \tau_{\min})$$

$$\text{that } \tau_{\max} = 1050 \text{ and } \tau_{\min} = 0$$

$$\text{also } R_T(t) = T_{\min} + 2/3 (T_{\max} - T_{\min}) (0,75 + t) \text{ where}$$

$$T_{\max} = 27,306666673$$

$$T_{\min} = 21,573333333$$

finally the weights and biases are

$$w_1^{(1)} = 11,182785114029016$$

$$w_1^{(2)} = 6,949537070178358$$

$$w_2^{(1)} = 7,573209499277103$$

$$w_2^{(2)} = 0,8310318767165362$$

$$w_3^{(1)} = 13,363060428561267$$

$$w_3^{(2)} = -1,0052120737394616$$

$$\theta_1^{(1)} = -8,128164020909667$$

$$\theta_1^{(2)} = 3,4487370964491277$$

$$\theta_2^{(1)} = -2,4483745545162416$$

$$\theta_3^{(1)} = -8,307475498170888$$

All these weights and biases are coded as 64 bits long genes. For a better learning 100 genomes are used. All these genomes undergo the crossover operation (multi-crossover) in which the bits from one genome is inherited from the mate with a particular probability, i.e. several bits in a genome can be changed after the crossover. Then the mutation operation is applied to the offspring genomes with specified probability (mutation probability) the particular bits in all processed genomes are inverted. Then every 90 best offspring genomes plus 10 best parent genomes make a new population of 100 genomes. The selection of 100 best genomes is done on the basis of the genome fitness value [13].

GA is used with multi-point crossover with probability of 0,25, mutation probability of 0,05 and size of population 100.

The learning may be stopped once we see the process not showing any significant change in fitness value during many succeeding regenerations. In this case what we can do is to specify new mutation (and maybe crossover) probability and continue the process. If the obtained total error performance index or the behavior of the obtained network is not desired, we can restructure the network by adding new hidden neurons, or do better sampling (fuzzification) of the learning patterns [13].

5.2. Investigation of Accuracy NFG Model of NiCd Battery

The comparison of the proposed models with actual voltage and temperature are shown in Figure 5.2 and Figure 5.3 respectively.

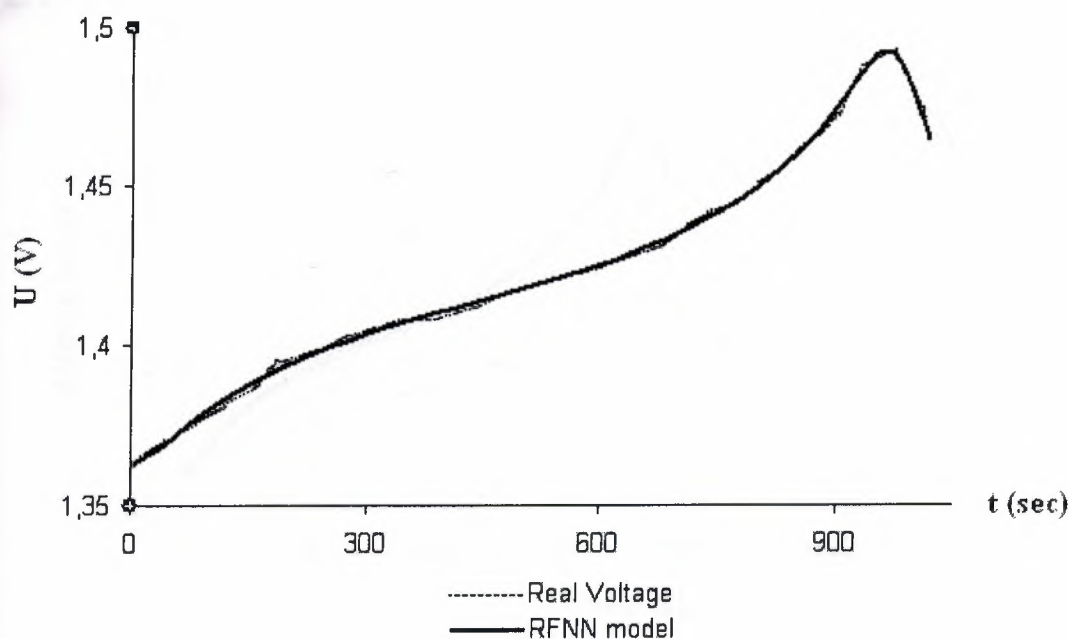


Figure 5.2 Voltage model with Actual Voltage

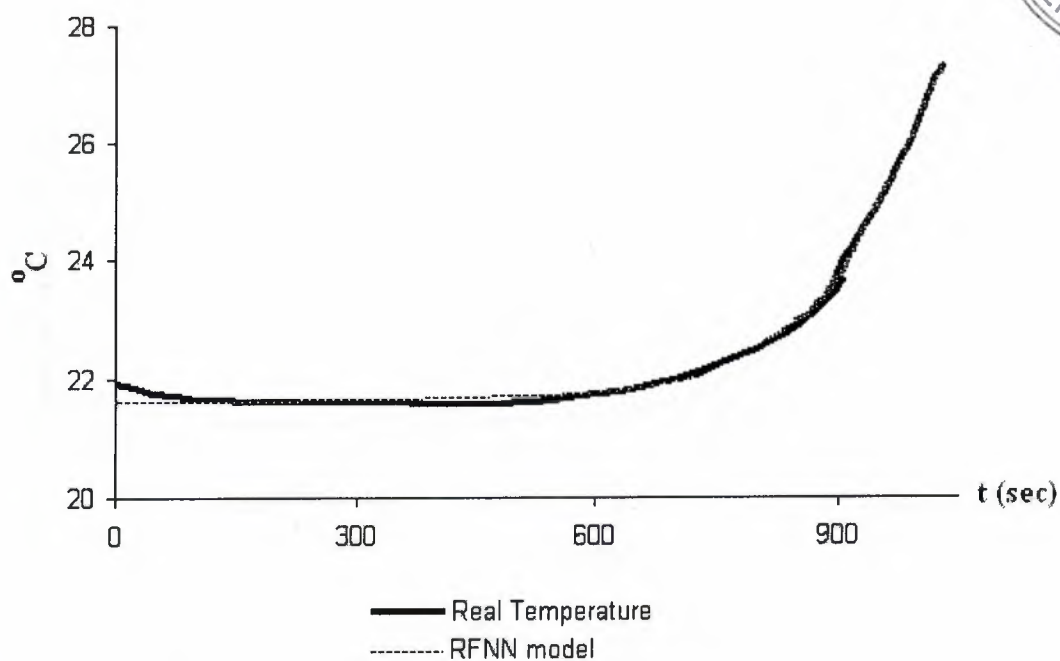


Figure 5.3 Temperature model with Actual Temperature

The system designs the voltage and temperature characteristics for the different C-rates as shown in Figure 5.4 and Figure 5.5.

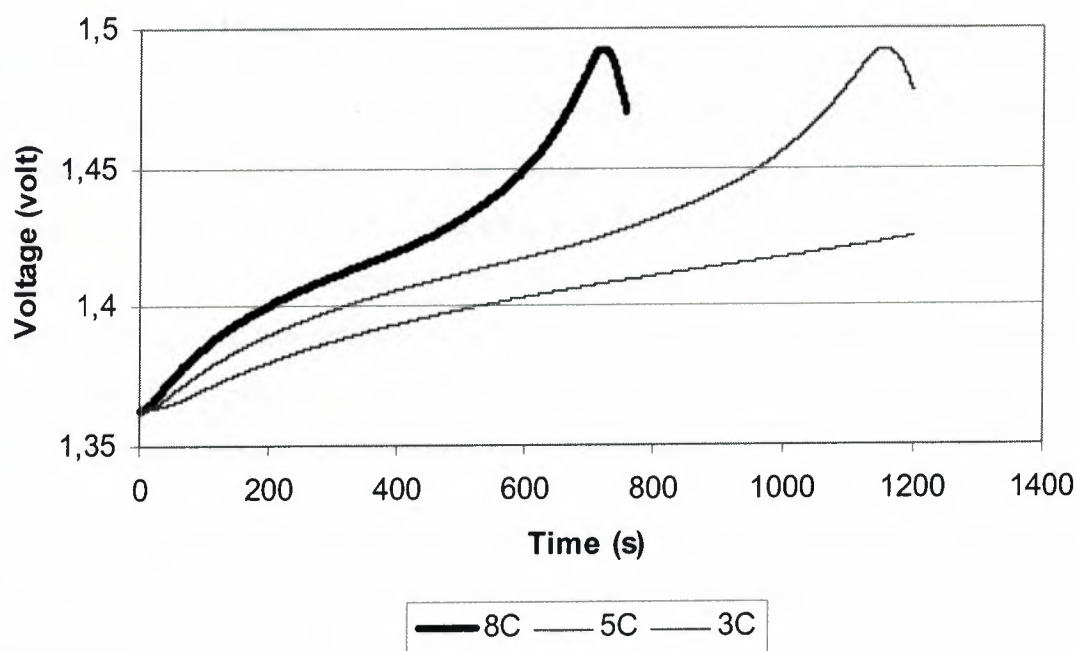


Figure 5.4 Voltage characteristics for 8C, 5C and 3C

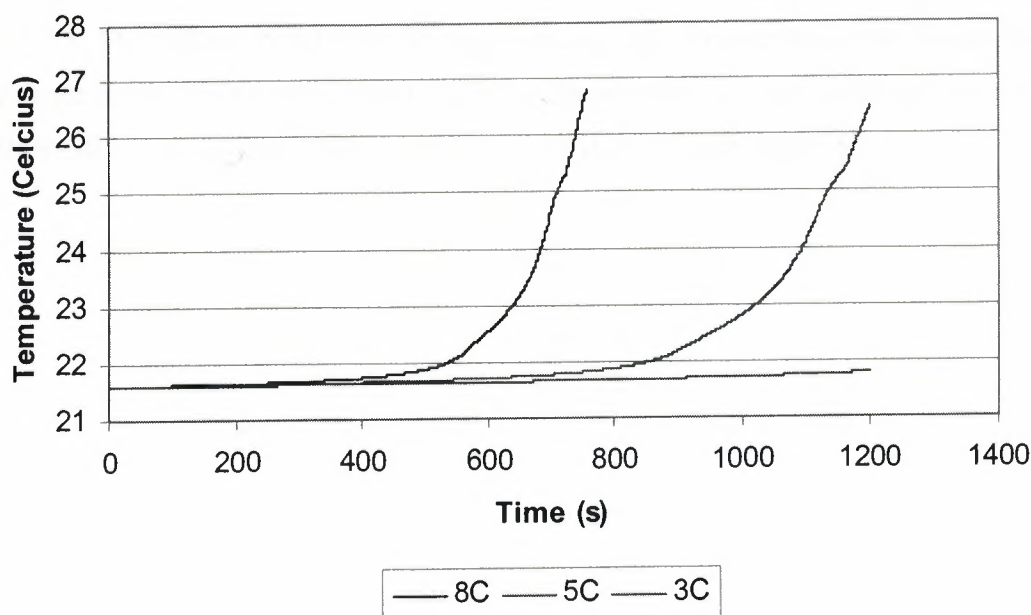


Figure 5.5 Temperature characteristics for 8C, 5C and 3C

5.3 Software Development Neuro-Fuzzy-Genetic Modeling of NiCd Batteries Charging

To design a model for Nickel Cadmium voltage and temperature characteristics during battery charging process a Neural Network Simulator (NSIM) with Java programming language implementation has been used. The program is used to train the RFNN with Genetic Algorithm. Two C#.NET class programs designed to create an identifier neural network program. It has 4 inputs, 20 hidden and 1 outputs. The neural networks that represent voltage and temperature have 3 input 10 hidden and 1 output layer structures respectively. The input-output data used by these networks are stored in two files. one holds the voltage data and the other holds the temperature.

To use and compile these programs, two dll and two bat files are created. The first bat file compiles the C#.NET program for battery and the other one compiles the C#.NET program for genetic algorithm.

The results of NSIM program copied into an MS Excel file for temperature and voltage modeling. The temperature MS Excel file uses a macro program written with Visual Basic to evaluate the data in temperature data file using temperature network. The

results of the temperature model for different charging rates are shown in Figure 5.2 and Figure 5.4. The voltage MS Excel file uses a macro program written with Visual Basic to evaluate the data in voltage data file using voltage network. The results of the voltage model for different charging rates are shown in Figure 5.1 and Figure 5.3.

6. EXPERIMENTAL INVESTIGATION OF INTELLIGENT NEURO-FUZZY-GENETIC CONTROL SYSTEM FOR NICD BATTERY

6.1 Dynamic Data Mining Technique for Battery Charging Rules

Extraction

Battery charging controllers design and application is a growing industry direction. Fast and efficient charging of battery packs is a problem which is difficult and often expensive to solve using conventional techniques. The majority of existing works on intelligent charging systems are based on expert knowledge and heuristics. Not all features of the desired charging behavior can be attained by the hard-wired logic implemented by expert generated rules. Because the battery charging is a highly dynamic process and the chemical technology a battery uses varies significantly for different battery types, data mining technique can be of real importance for extracting the charging rules from the large databases, especially, when the charging logic is to be continuously changed during the life of the battery dependent on the type and characteristics of the battery and utilization conditions. Soft Computing Based Data Mining technique used for extraction of control rules for effective and fast battery charging process. The obtained rules were used for NiCd battery charging.

Neural networks are one of the techniques widely used today for data mining from unknown systems and creating self-adaptable intelligent controllers. One of the indictments against crisp neural networks is that it is difficult to understand the model that they have built and also how the raw data affects the output. The complex models of the neural network are captured solely by the link weights in the network which represent a very complex mathematical equation. There have been several attempts to alleviate these basic problems of the neural network. The simplest approach is to actually look at the neural network and try to create plausible explanations for the meanings of the hidden nodes. For example, paper [43] describes the ways for extracting the embedded knowledge in trained crisp neural networks in form of

symbolic rules. The rules in symbolic form are necessary for human interpretation, verification, and adjustment to acknowledge the trust level to the mined rule base.

To design the required intelligent charger, there is a need for extraction of efficient rule base. The rule base should be flexible and should be adjusted on-line during the life of the battery. These updates the most effectively can be done by a trainable Soft Computing based system. However, as it was mentioned in literature, the fuzzy neural networks will allow a more natural, complete, and transparent rule generation than in case of crisp neural networks [32]. Moreover, the highly dynamic chemical processes in battery would require application of recurrent neural networks proven to be more efficient in such cases.

Taking into consideration the above reasons we have come to the conclusion that for the battery (which is a highly dynamic object with mathematically undetermined and changeable model) the charging process can most efficiently be managed by using fuzzy inference system (FIS) with the rule base extracted from the Data Mining System implemented by a Fuzzy Recurrent Neural Network (RFNN). The necessary behavior is obtained by the RFNN after the training on the basis of experimental data. Then the charging control rules are extracted from the RFNN for using in the FIS-based charging controller.

A soft computing approach proposed based on a fuzzy recurrent neural network trained by genetic algorithms to generate a fuzzy rule base to control battery charging process. Rule extraction is one of the major forms of data mining and is perhaps the most common form of knowledge discovery in learning systems [32-34]. Fuzzy neural networks, which use fuzzy inputs, fuzzy connection weights, and fuzzy outputs, can be converted to fuzzy If-Then rules directly, as they use limited number of linguistic terms for each input and output variables. The data mining process for the If-Then rules extraction for battery charging consists of the following steps:

1. Collecting the detailed data base. Since data mining relies heavily on training data, it is important to collect data very carefully. Collected data base is cleansed and filtered.

2. Development of the model structure. The structure is selected on the basis of available data and expected complexity of the model. For battery charging controller we used 4 input variables (feeding RFNN input neurons) and 1 output variable (RFNN output neuron). Number of hidden neurons is varying and is part of optimization procedure.
3. Fuzzification. Defining term sets for input and output variables. The initial terms are set to represent equally shaped and uniformly distributed membership functions. They will fuzzify the data entering the network.
4. Training neural network and obtaining initial input-output relationships by GA based technique. Obtaining network weights.
5. Adjustment of membership functions of used terms by GA.
6. Extracting the rules and human analysis.
7. Optimization of the number of rules and terms. Previous steps are repeated, if necessary. This optimization is done using GA having considered the required accuracy, existing complexity and performance.
8. Fuzzy control system update. The fuzzy rule base extracted from the trained and optimized RFNN is incorporated into the intelligent battery charging control FIS.
9. Loop to Step 1. The dynamic nature of the charging process requires continuous data mining process updating the extracted and incorporated control rules.

The battery charging control rules obtained as a result of the used data mining technique are listed in Table 6.1.

Table 6.1 The control rules

T	dT/dt	U	dU/dt	I
LOW	LOW	HIGH	LOW	LOW
LOW	LOW	HIGH	MEDIUM	LOW
LOW	LOW	HIGH	HIGH	LOW
LOW	MEDIUM	HIGH	LOW	LOW
LOW	HIGH	HIGH	LOW	LOW
LOW	MEDIUM	HIGH	MEDIUM	LOW

LOW	HIGH	HIGH	MEDIUM	LOW
LOW	MEDIUM	HIGH	HIGH	LOW
LOW	HIGH	HIGH	HIGH	LOW
MEDIUM	LOW	HIGH	LOW	LOW
MEDIUM	LOW	HIGH	MEDIUM	LOW
MEDIUM	LOW	HIGH	HIGH	LOW
MEDIUM	MEDIUM	HIGH	LOW	LOW
MEDIUM	HIGH	HIGH	LOW	LOW
MEDIUM	MEDIUM	HIGH	MEDIUM	LOW
MEDIUM	HIGH	HIGH	MEDIUM	LOW
MEDIUM	MEDIUM	HIGH	HIGH	LOW
MEDIUM	HIGH	HIGH	HIGH	LOW
HIGH	LOW	HIGH	LOW	LOW
HIGH	LOW	HIGH	MEDIUM	LOW
HIGH	LOW	HIGH	HIGH	LOW
HIGH	MEDIUM	HIGH	LOW	LOW
HIGH	HIGH	HIGH	LOW	LOW
HIGH	MEDIUM	HIGH	MEDIUM	LOW
HIGH	HIGH	HIGH	MEDIUM	LOW
HIGH	MEDIUM	HIGH	HIGH	LOW
HIGH	HIGH	HIGH	HIGH	LOW
LOW	LOW	MEDIUM	LOW	HIGH
LOW	LOW	MEDIUM	MEDIUM	HIGH
LOW	LOW	MEDIUM	HIGH	HIGH
LOW	MEDIUM	MEDIUM	LOW	HIGH
LOW	HIGH	MEDIUM	LOW	HIGH
LOW	MEDIUM	MEDIUM	MEDIUM	HIGH
LOW	HIGH	MEDIUM	MEDIUM	HIGH
LOW	MEDIUM	MEDIUM	HIGH	HIGH
LOW	HIGH	MEDIUM	HIGH	HIGH
MEDIUM	LOW	MEDIUM	LOW	HIGH
MEDIUM	LOW	MEDIUM	MEDIUM	HIGH

MEDIUM	LOW	MEDIUM	HIGH	HIGH
MEDIUM	MEDIUM	MEDIUM	LOW	HIGH
MEDIUM	HIGH	MEDIUM	LOW	HIGH
MEDIUM	MEDIUM	MEDIUM	MEDIUM	HIGH
MEDIUM	HIGH	MEDIUM	MEDIUM	HIGH
MEDIUM	MEDIUM	MEDIUM	HIGH	MEDIUM HIGH
MEDIUM	HIGH	MEDIUM	HIGH	MEDIUM LOW
HIGH	LOW	MEDIUM	LOW	MEDIUM LOW
HIGH	LOW	MEDIUM	MEDIUM	LOW
HIGH	LOW	MEDIUM	HIGH	LOW
HIGH	MEDIUM	MEDIUM	LOW	LOW
HIGH	HIGH	MEDIUM	LOW	LOW
HIGH	MEDIUM	MEDIUM	MEDIUM	LOW
HIGH	HIGH	MEDIUM	MEDIUM	LOW
HIGH	MEDIUM	MEDIUM	HIGH	LOW
HIGH	HIGH	MEDIUM	HIGH	LOW
LOW	LOW	LOW	LOW	LOW
LOW	LOW	LOW	MEDIUM	LOW
LOW	LOW	LOW	HIGH	LOW
LOW	MEDIUM	LOW	LOW	LOW
LOW	HIGH	LOW	LOW	LOW
LOW	MEDIUM	LOW	MEDIUM	LOW
LOW	HIGH	LOW	MEDIUM	LOW
LOW	MEDIUM	LOW	HIGH	LOW
LOW	HIGH	LOW	HIGH	LOW
MEDIUM	LOW	LOW	LOW	LOW
MEDIUM	LOW	LOW	MEDIUM	LOW
MEDIUM	LOW	LOW	HIGH	LOW
MEDIUM	MEDIUM	LOW	LOW	LOW
MEDIUM	HIGH	LOW	LOW	LOW
MEDIUM	MEDIUM	LOW	MEDIUM	LOW
MEDIUM	HIGH	LOW	MEDIUM	LOW

MEDIUM	MEDIUM	LOW	HIGH	LOW
MEDIUM	HIGH	LOW	HIGH	LOW
HIGH	LOW	LOW	LOW	LOW
HIGH	LOW	LOW	MEDIUM	LOW
HIGH	LOW	LOW	HIGH	LOW
HIGH	MEDIUM	LOW	LOW	LOW
HIGH	HIGH	LOW	LOW	LOW
HIGH	MEDIUM	LOW	MEDIUM	LOW
HIGH	HIGH	LOW	MEDIUM	LOW
HIGH	MEDIUM	LOW	HIGH	LOW
HIGH	HIGH	LOW	HIGH	LOW

All weights and biases of the RFNN are coded as 64 bits long genes. For a better learning we use 100 genomes. All these genomes undergo the crossover operation (multi-crossover) [15] in which the bits from one genome are inherited from the mate with a particular probability, i.e. several bits in a genome can be changed after the crossover. Then the mutation operation is applied to the offspring genomes with specified probability (mutation probability) the particular bits in all processed genomes are inverted. Then every 90 best offspring genomes plus 10 best parent genomes make a new population of 100 genomes. The selection of 100 best genomes is done on the basis of the genome fitness value [15]. GA used with multi-point crossover with probability 0,25, mutation probability 0,05 and size of population 100.

The FRRN designed for extracting the fuzzy control rules had 4 inputs, 20 hidden neurons, and 1 output. The three used inputs represented temperature (T), change of temperature (dT), voltage (V) and change of voltage (dV). The output of the controller is the current (I) applied for charging the battery.

The network was trained by experimental data obtained from a large data base produced from readings collected from different sensors and further purified. Figure 6.1 shows the graph of the charging process under the fuzzy logic based charging controller with the extracted rule base. In majority of simulation experiments, GA learning was done

with population size 100, probability of multi-point crossover 0.25, and probability of mutation 0.05.

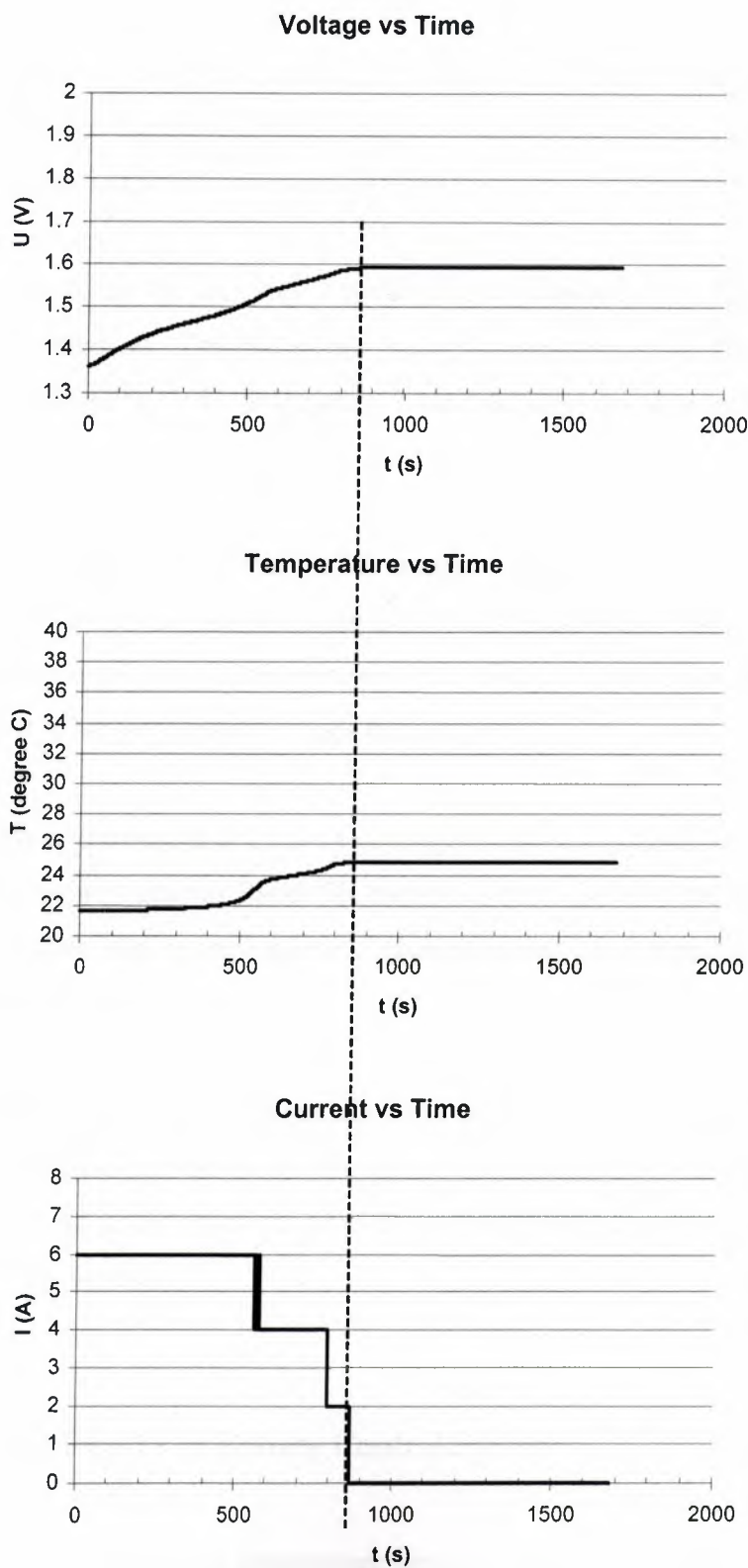


Figure 6.1 Battery charging control process

6.2 Software Development of Neuro-Fuzzy-Genetic Control System for NiCd Batteries Charging

To design a controller for NiCd battery charging process a Neural Network Simulator for crisp weights and signals with Java implementation, the temperature and voltage models that explained in chapter five has been used.

The rules trained to the system with TrainRules.bat (Appendix A.12) program. This program is compiled as TrainRules.exe with the reference of Network.dll and TrainRules.cs (Appendix A.5) programs. TrainRules.cs uses the rules.io and the rules.nw.

FR_ChargingControl.bat (Appendix 1.14) creates the CCont.exe program from Battery.dll, Network.dll and FR_ChargingControl.cs (Appendix 1.4) program. This program gives the current output, temperature and voltage values in seconds according to the models. FR_ChargingControl.cs uses the rules.nw.

CCont.bat file stores the output of CCont.exe program into res.txt file. From this file the numerical values copied into an Excel file named as test.xls, then the temperature vs time, voltage vs time and current vs time characteristics of the designed controller converted to graphic formats.

CCont.bat file stores the output of CCont.exe program into res.txt file. From this file the numerical values copied into an Excel file named as test.xls, then the temperature vs time, voltage vs time and current vs time characteristics of the designed controller converted to graphic formats.

6.3 Experimental Results of Battery Control System

Figure 6.2 shows the voltage and temperature graphs, when we first apply 6A for 600 seconds and then 2A until we reach the required charge of 1.6V. Figure 6.3 shows the

curves displaying the derivative (speed of change) of voltage and temperature. Right to the graph, the trapezoid membership functions for terms used in fuzzy control rules as values for the variables dU , change of voltage, and dT , change of temperature, are shown: see left and right sides, respectively.

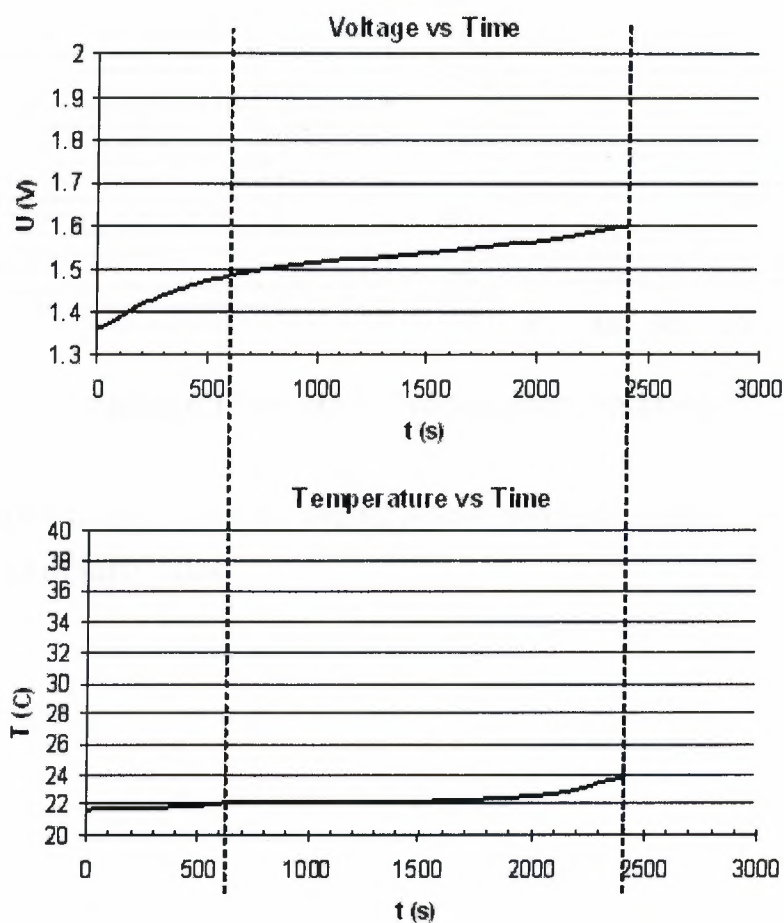


Figure 6.2 Voltage and temperature

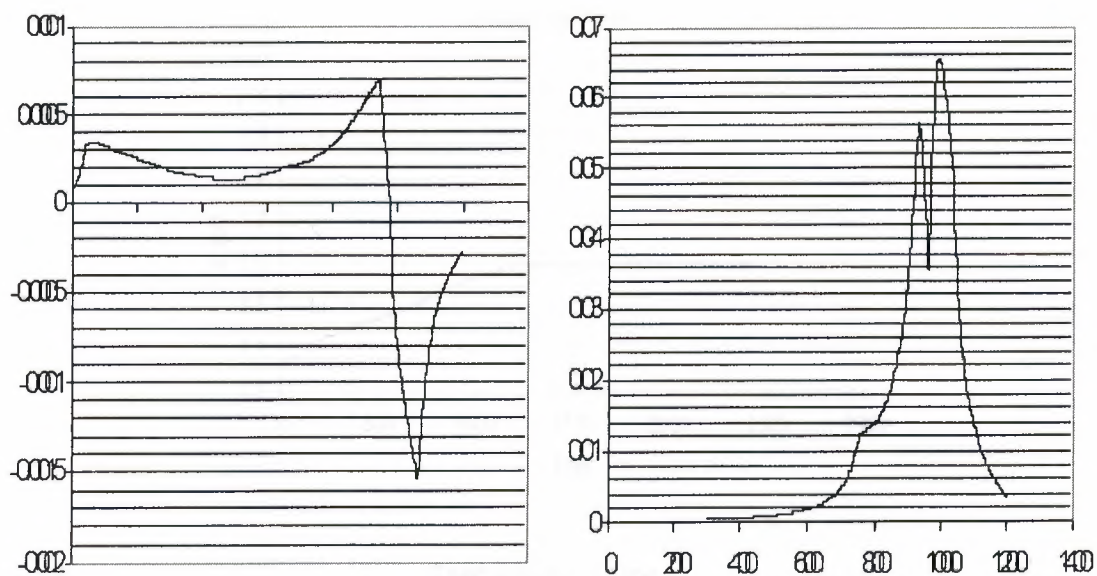


Figure 6.3 Derivative of voltage and temperature

Figure 6.4 shows the graph of the charging process under the control of neural network learned by a set of fuzzy rules.

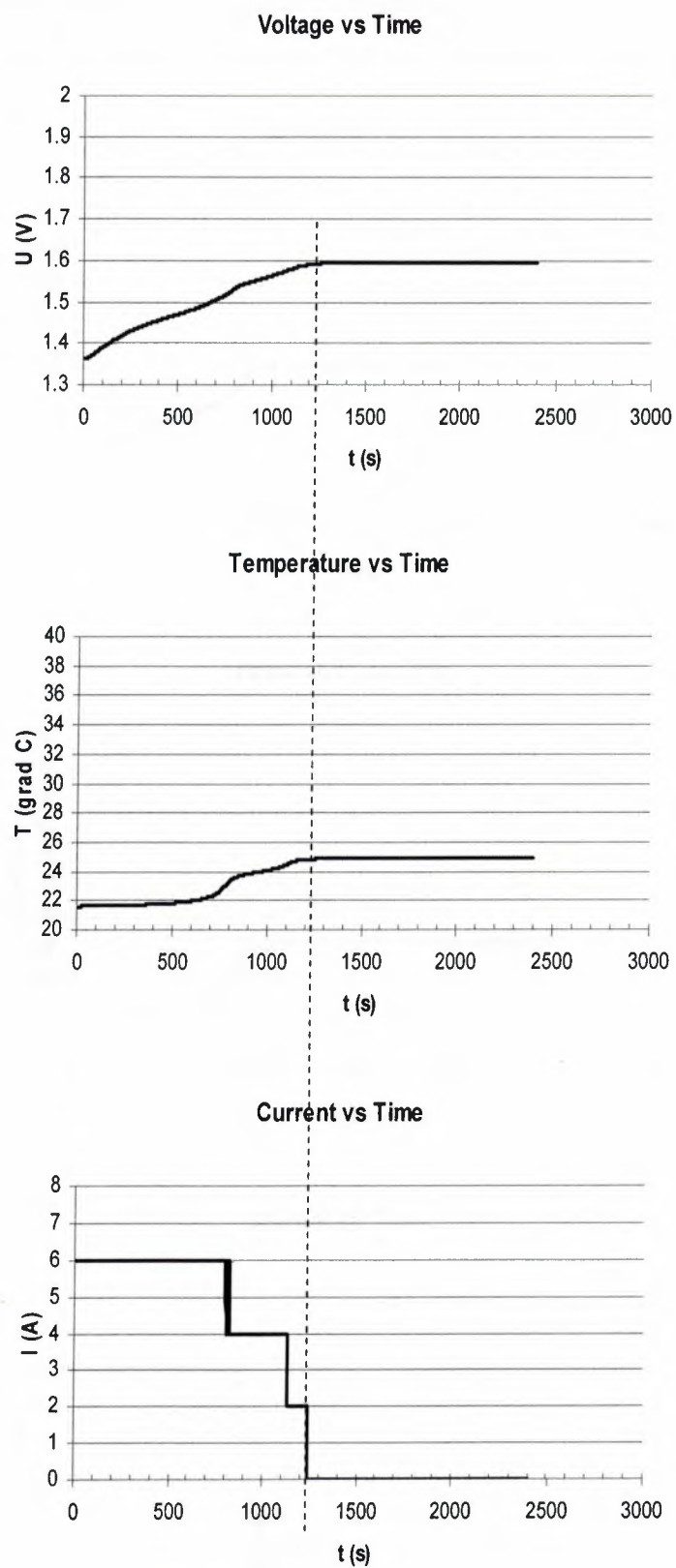


Figure 6.4 Graph of the charging process

Figure 6.5 shows the control with modified temperature term membership functions to allow slightly higher temperature. This has allowed a decrease in the charging time.

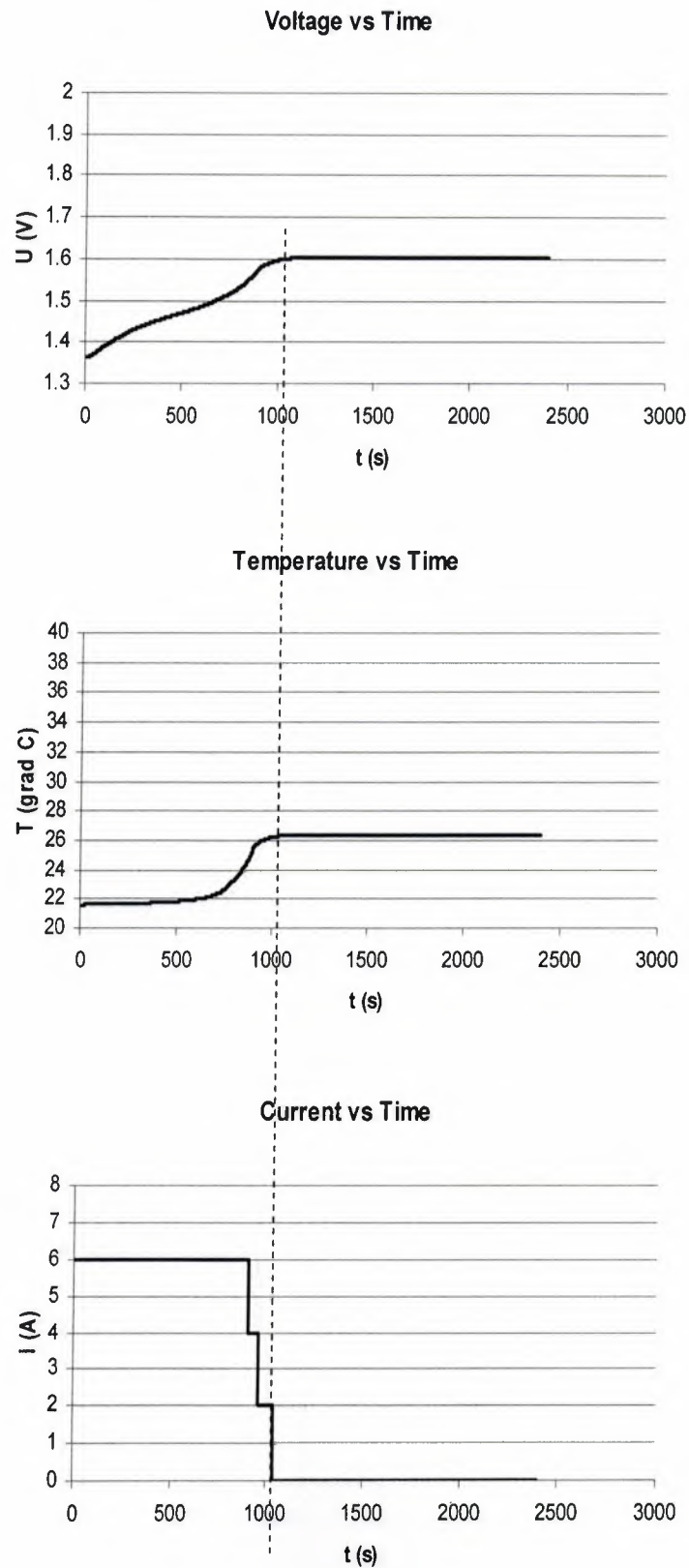


Figure 6.5 The control with modified temperature term membership functions

Figures 6.6 and Figure 6.7 show cases when we intentionally remove some rules from consideration in control system.

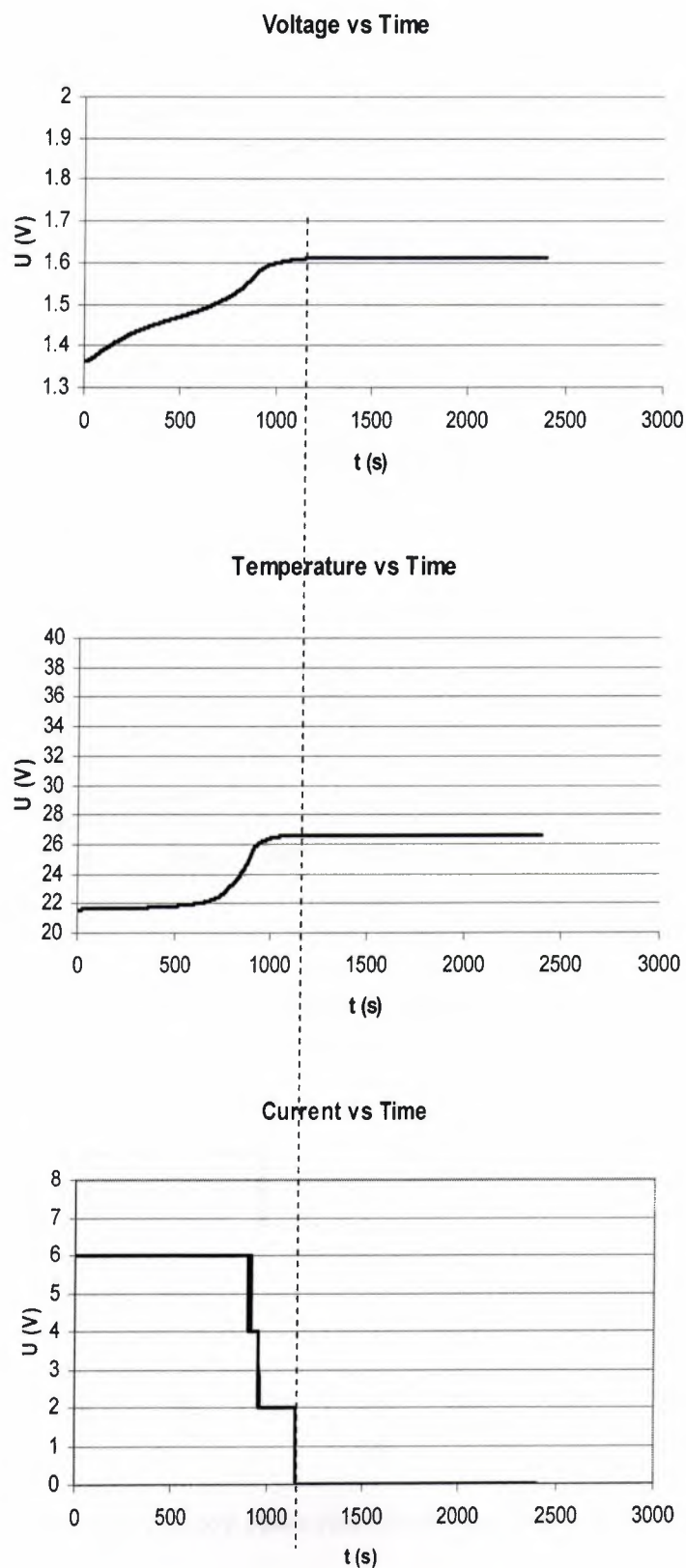


Figure 6.6 Intentionally remove some rules from consideration in control system.

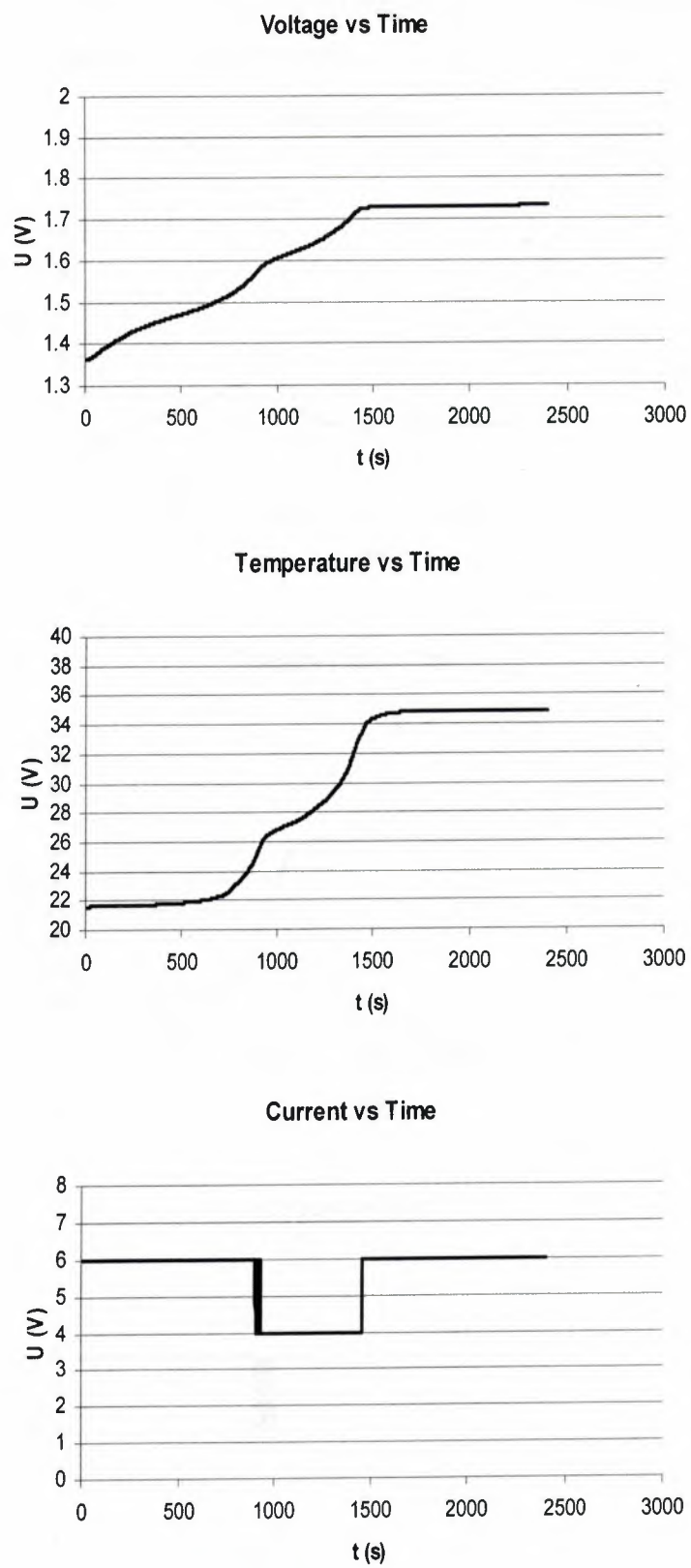


Figure 6.7 Intentionally remove some rules from consideration in control system

Figure 6.8 shows the case when we change (corrupt) membership functions of terms for the variables U (voltage) and T (temperature)

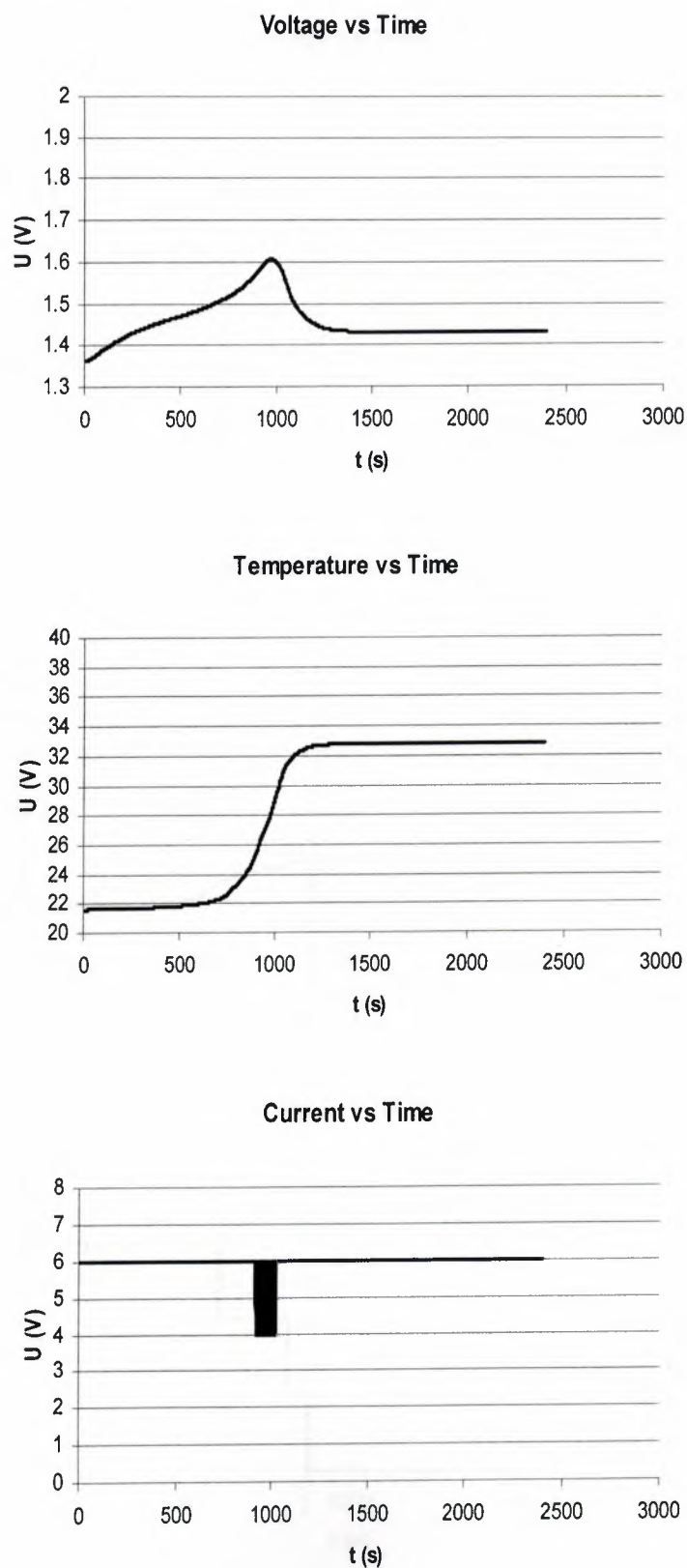


Figure 6.8 Corrupt membership functions of terms for the variables U and T

Other experiment been done to decrease the $T_{end}-T_{start}$ result and charging time. The best results of the neuro-fuzzy-genetic controller are shown in Figure 6.9.

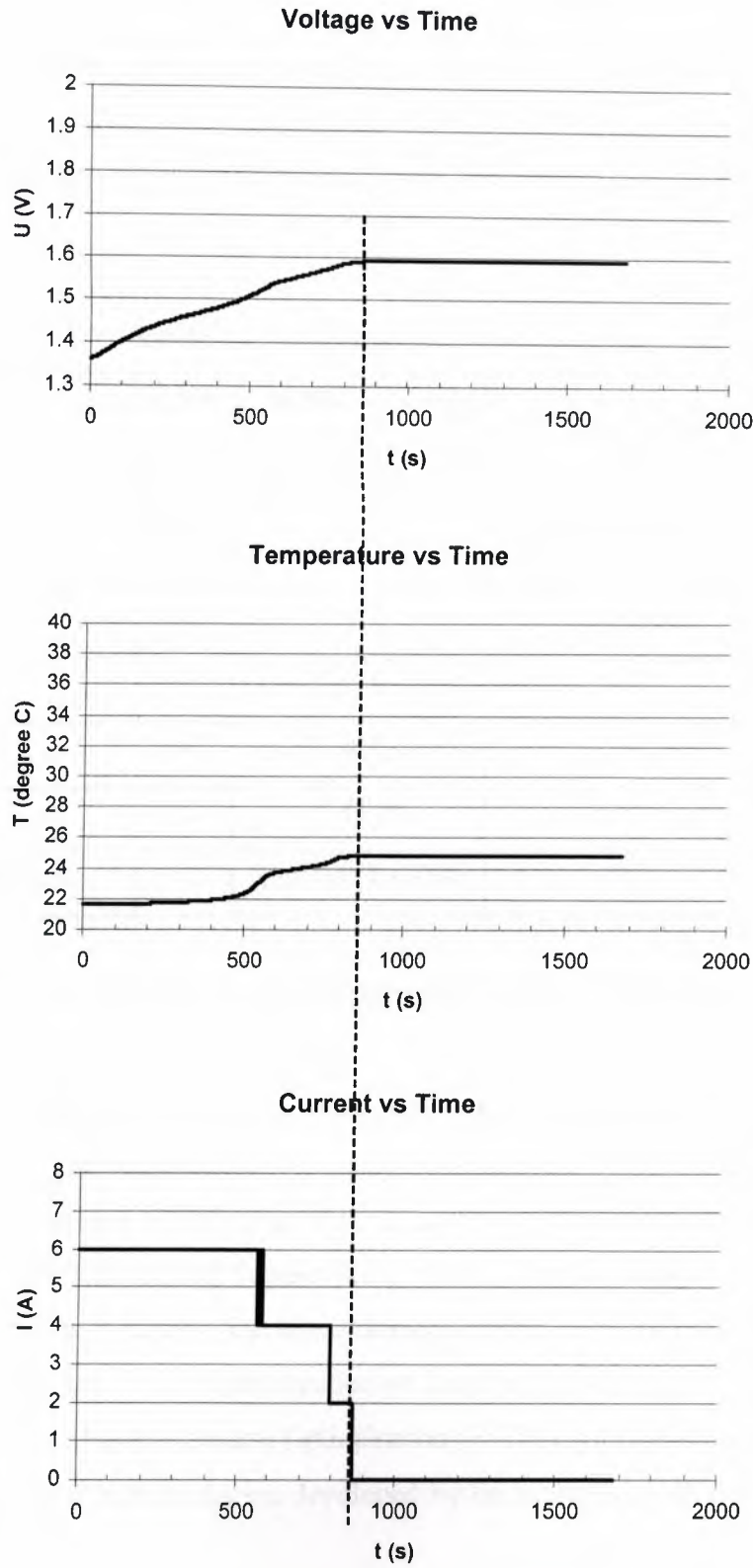


Figure 6.9 Results of a Charging process

6.4 Compatible Analysis of Neuro-Fuzzy-Genetic Control System for Batteries Charging

The results of proposed charging controller compared with other battery chargers are given in Table 6.2. The NiCd charger with the rule base extracted on the basis of the developed technique gives less charging time and less $T_{\text{end}}-T_{\text{start}}$ result than other controllers.

Table 6.2 Comparison of NiCd chargers

	Max C-rate	Charge Control Inputs	Time (sec)	$T_{\text{end}}-T_{\text{start}}$ (°C)
Conventional Fast Charger [2]	1C	U,T	3600-5400	5
NeuFuz [2]	4C	U,T	1200-1800	5
FG [3]	6C	U,dU/dt,T,dT/dt	959	9
FL [7]	6C	U,dU/dt,T,dT/dt	-	35-60
ANFIS [9]	8C	T,dT/dt	900	50
Proposed approach	6C	U,dU/dt,T,dT/dt	860	2,85

6.5 Identification of the Proposed Controller for NiCd Batteries Charger

There are some intelligent works on NiCd batteries charging controllers identification [2], [3] and [9]. The identification of fuzzy controller for NiCd batteries charger by applying fuzzy c-means (FCM) clustering algorithm on the input-output training data proposed in [9]. The Takagi-Sugeno-Kang (TSK) model obtained through FCM clustering algorithm is further fine tuned through hybrid learning in [35]. The authors presents a framework for the identification of fuzzy models from the available input-output data through Particle Swarm Optimization (PSO) algorithm in [36] that the data from the rapid NiCd battery charger developed by the authors in [9] and [35] has been used.

In this section, brief information is given on dynamic system identification using RFNN with back propagation learning algorithm. A four-layer RFNN, as in [37] which is comprised of an input layer, a membership layer, a rule layer, and an output layer, is adopted to implement the proposed RFNN. The charger to be identified is dynamic system whose outputs are functions of past inputs and past outputs as well. For this dynamic system identification, since a recurrent network the TRFN is used, only the current state of the system and control signal are fed as input to the network. The adopted identification configuration is a serial-parallel model from [37] shown in Figure 6.10. The $y(k)$ is the desired output, $\hat{y}(k)$ is the current output, $u(k)$ is control input and $E(k)$ is error function.

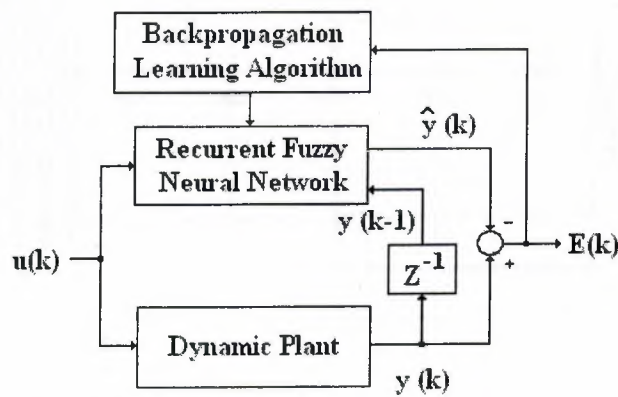


Figure 6.10 Dynamic modeling of nonlinear systems using RFNN

The nonlinear plant with multiple time delay is described as

$$\hat{y}(k) = f(y(k), y(k-1), y(k-2), u(k), u(k-1)) \quad (6.1)$$

where

$$f(x_1, x_2, x_3, x_4, x_5) = \frac{x_1 x_2 x_3 x_5 (x_3 - 1) + x_4}{1 + x_2^2 + x_3^2} \quad (6.2)$$

Here, the current output of the plant depends on three previous outputs and two previous inputs. For the testing signal $u(k)$ the following equation was used:

$$u(k) = \begin{cases} \sin\left(\frac{\pi k}{25}\right), & 0 < k < 250 \\ 1, & 250 \leq k < 500 \\ -1, & 500 \leq k < 750 \\ 0.3 \sin\left(\frac{\pi k}{25}\right) + 0.1 \sin\left(\frac{\pi k}{32}\right) + 0.6 \left(\frac{\pi k}{25}\right), & 750 \leq k < 1000 \end{cases} \quad (6.3)$$

Mean square error (MSE) is defined as

$$\text{MSE} = \frac{1}{N} \sum_{k=1}^N (y(k) - \hat{y}(k))^2 \quad (6.4)$$

where the N is the number of data points taken for model validation.

The simulation demonstrates the identification error (MSE) as 0,003267655 where the N is equal to 2000 is shown in Figure 6.11.

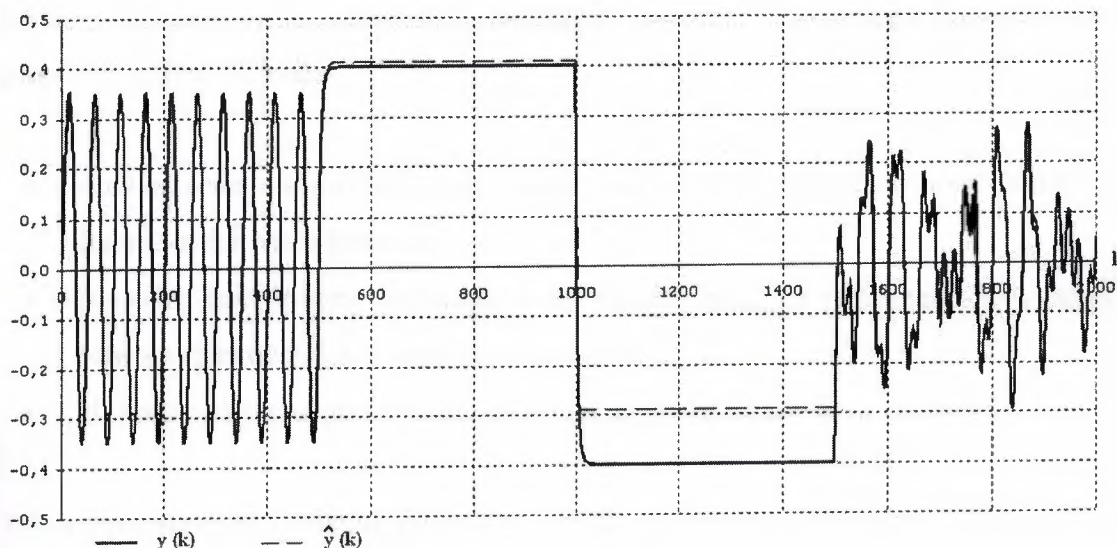


Figure 6.11 The output of the plant ($y(k)$) and the output of RFNN ($\hat{y}(k)$)

The comparison of the error values between the proposed and the existing intelligent controllers shown in Table 6.3.

Table 6.3 Comparison of the intelligent controllers MSE.

Charger	MSE
ANFIS [9]	0,132100000
ANFIS [35] (fuzzy c-means clustering algorithm applied on the input-output training data)	0,008979000
Mamdani model through Particle Swarm Optimization (PSO) algorithm [36]	0,145500000
Singleton model through Particle Swarm Optimization (PSO) algorithm [36]	0,112300000
Proposed approach	0,003267655

From the comparison of the MSE results it can be seen that proposed approach achieved the highest identification accuracy than the compared networks.

Conclusion

This thesis has presented a novel approach to modeling and controlling NiCd battery charging process. The proposed neuro-fuzzy-genetic model and controller system for NiCd batteries offers a unique combination of attributes intended to occupy a niche between the capabilities of existing intelligent chargers. The unique features of the proposed system include:

- A novel genetic algorithm based trained recurrent fuzzy neural network model and controller for NiCd batteries.
- Dynamic data mining technique used on the basis of recurrent fuzzy neural network to extract fuzzy rules for the controller.
- Minimize both charging time and internal temperature increase.

The comparisons with existing systems show that the suggested approach gives the least identification error besides least charging time and least internal temperature increase during charging. Proposed approach also detects the difference between deeply discharged and shorted cell batteries.

Further work will be carried out to investigate the design of a neuro-fuzzy-genetic model and controller for quickly charging Nickel Metal Hydride (NiMH) batteries. Although the Nickel Cadmium is the most popular battery type used in space, NiMH batteries are currently being used to power the aerospace systems such as Hubble telescope, International Space Station (besides NiCd) and being considered for incorporation in further projects like SELENE (SELenological and ENgineering Explorer, Launch Date: 2007-07-01). Applications of NiMH type batteries include hybrid vehicles such as the Toyota Prius, humanoid prototype robot ASIMO designed by Honda and consumer electronics.

References

- [1] O. Castillo, P. Melin, "Soft Computing for Control of Non-Linear Dynamical Systems", Springer, Germany, 2001.
- [2] M. Z. Ullah, B. Burford, S. Dilip, "Method and Apparatus for Fast Battery Charging using Neural Network Fuzzy Logic Based Control", IEEE Aerospace and Electronic Systems Magazine, vol. 11, issue. 6, pp. 26-34, 1996.
- [3] H. Surmann, "Genetic Optimization of a Fuzzy System for Charging Batteries", IEEE Trans. on Industrial Electronics, vol. 43, no. 5, pp. 541-548, Oct. 1996.
- [4] Sanyo, CADNICA Batteries, Sanyo Electric Co. Ltd, 2002.
- [5] Panasonic, Nickel Cadmium Handbook, 2003.
- [6] Energizer, Nickel Cadmium Batteries Application Manual, 2001.
- [7] P.D. Ionescu, M. Moscalu, A. Moscalu, "Intelligent charger with fuzzy logic", Proceedings of Int. Symp. on Signals, Circuits and Systems 2003 (SCS 2003), vol. 1, pp. 101-104, July 2003.
- [8] D. VonderHaar, "Conductance Controlled Charging", 113th Convention Battery Council International, Las Vegas, Nevada, May 5, 2001.
- [9] A. Khosla, S. Kumar, K. K. Aggarwal, "Fuzzy controller for rapid nickel-cadmium batteries charger through adaptive neuro-fuzzy inference system (ANFIS) architecture", 22nd International Conference of the North American Fuzzy Information Processing Society, NAFIPS 2003, pp. 540 – 544, July 24-26, 2003.

- [10] J. Diaz, J.A. Martin-Ramos, A.M. Pernia, F. Nuno, F.F. Linera, "Intelligent and universal Fast Charger for Ni-Cd and Ni-MH Batteries in portable Applications", IEEE Trans. on Industrial Electronics, vol. 51, issue. 4, pp. 857-863, Aug. 2004.
- [11] A. Mukherjee, "Advances in Battery Management Using Neural Networks and Fuzzy Logic", MSc. Thesis, School of ECE, Cornell University, NY, 2003.
- [12] R.A. Aliev, B. Fazlollahi, R.M. Vahidov, "Genetic algorithm-based learning of fuzzy neural networks. Part 1: feed-forward fuzzy neural networks", J. Fuzzy Sets and Systems 118, pp. 351-358, 2001.
- [13] R.A. Aliev, B. Fazlollahi, R.R. Aliev, B. Guirimov, "Genetic algorithm-based learning of fuzzy neural networks. Part 2: Recurrent Fuzzy Neural Networks", to be published.
- [14] R.A. Aliev, B. Fazlollahi, R.M. Vahidov, "Artificial Neural Networks. Theory and practice", Tabriz University Press, Tabriz, 1995.
- [15] R.A. Aliev, R.R. Aliev, "Soft Computing and It's Applications", World Scientific, NJ, London, Singapore, Hong Kong, 2001.
- [16] R.A. Aliev, G.A. Mamedova, R.R. Aliev, "Fuzzy sets Theory and its Application", Tabriz University, Tabriz, 1993.
- [17] R.A. Aliev, K.W. Bonfig, U.K. Shirinova, "Genetic Algorithm based fuzzy modeling.", Proceeding of the 3rd International Conference on Application Fuzzy Systems and Soft Computing-ICAFFS'98, pp. 118-123, Wiesbaden, Germany, 1998.
- [18] L.A. Zadeh, "Fuzzy Sets", J. Information and Control 8, pp. 338-353, 1965.

- [19] L.A. Zadeh, "Fuzzy Logic", J. IEEE Computer, pp. 83-93, April 1988.
- [20] L.A. Zadeh, "Fuzzy Logic, Neural networks and soft computing", J. Comm of ACM 37 (3), pp. 77-84, 1994.
- [21] C.-H. Lee, C.-C. Teng, "Identification and Control of Dynamic Systems Using Recurrent Fuzzy Neural Networks", IEEE Trans. on Fuzzy Systems, vol. 8, no. 4, pp. 349-366, August 2000.
- [22] Y.-C. Wang, C.-J. Chien, C.-C. Teng, "Direct Adaptive Learning Control of Nonlinear Systems Using an Output-Recurrent Fuzzy Neural Network", IEEE Trans. on Systems, Man, and Cybernetics-Part B: Cybernetics, vol. 34, no. 3, pp. 1348-1359, June 2004.
- [23] J. Zhang, A.J. Morris, "Recurrent Neuro-Fuzzy Networks for Nonlinear Process Modeling", IEEE Trans. on Neural Networks, vol. 10, no. 2, pp. 313-326, March 1999.
- [24] P. Lui, H. Li, "Efficient Learning Algorithms for Three-Layer Regular Feed-forward Fuzzy Neural Networks", IEEE Trans. on Neural Networks, vol. 15, no. 3, pp. 545-558, May 2004.
- [25] C.-F. Juang, "A TSK-Type Recurrent Fuzzy Network for Dynamic Systems Processing by Neural Network and Genetic Algorithms", IEEE Trans. on Fuzzy Systems, vol. 10, no. 2, pp. 155-170, April 2002.
- [26] G. Bortolon, "An Architecture of Fuzzy Neural Networks for Linguistic Processing", Fuzzy Set and Systems 100, pp. 197-215, 1998.
- [27] S.-F. Su, F.-Y.P. Yang, "On the Dynamic Modeling with Neural Fuzzy Networks", IEEE Trans. on Neural Networks, vol. 13, no. 6, pp. 1548-1553, Nov. 2002.

- [28] I. Buchmann, "Batteries in a Portable World: A Handbook on Rechargeable Batteries for Non-Engineers", Second Edition, ISBN 0-9682118-2-8 Cadex Electronics Inc., 2001.
- [29] National Semiconductor's, "Temperature Sensor Handbook", National Semiconductor's web page www.national.com.
- [30] ABB, "Voltage Sensors, Current Sensors", ABB's web page www.abb.com.
- [31] Wikipedia, the free encyclopedia, www.wikipedia.org.
- [32] S. Mitra, Y. Hayashi, "Neuro-fuzzy rule generation: survey in soft computing framework", IEEE Trans. Neural Networks, vol. 11, no. 3, pp. 748-768, 2000.
- [33] I. Guyon, S. Gunn, M. Nikraves, and L. Zadeh, "Feature Extraction, Foundations and Applications", editors: Springer, New York, 2004.
- [34] W. Pedrycz, "Associations and Rules in Data Mining: a Linkage Analysis", in Proceedings of the 2002 IEEE International Conference on Fuzzy Systems, pp. 867-871, 2002.
- [35] A. Khosla, , S. Kumar, , K. K. Aggarwal, "Identification of fuzzy controller for rapid Nickel-Cadmium batteries charger through fuzzy c-means clustering algorithm", Proc. of the 22nd Int. Conf. of the North American Fuzzy Information Processing Society, pp. 536-539, 2003.
- [36] A. Khosla, , S. Kumar, , K. K. Aggarwal, "A Framework for Identification of Fuzzy Models through Particle Swarm Optimization Algorithm", 2005 Annual IEEE INDICON, pp. 388-391, 2005.
- [37] Chih-Min Lin, Chun-Fei Hsu, "Identification of Dynamic Systems Using Recurrent Fuzzy Neural Network", Proceedings of the Joint 9th IFSA World

- Congress and 20th NAFIPS International Conference, vol. 5, pp. 2671-2675, 2001.
- [38] Y. Hayashi, J.J. Buckley, E. Czogala, "Fuzzy Neural Networks with Fuzzy Signals and Weights", *IJCNN'92*, vol. 2, pp. 697-701, 1992.
 - [39] M. Gupta, H. Ding, in: F. Aminzadeh, M. Jamshidi (Eds.), "Foundations of Fuzzy Neural Computations, Soft Computing: Fuzzy Logic, Neural Networks, and Distributed Artificial Intelligence", Prentice Hall, Englewood Cliffs, NJ, pp. 165-195, 1994.
 - [40] H. Ishibuchi, K. Morioka, H. Tanaka, "A Fuzzy Neural Network with Trapezoid Fuzzy Weights", *IEEE, New York, 1994, Proceedings of the Third IEEE Fuzzy Systems, IEEE World Congress on Computational Intelligence*, vol. 1, pp. 228-233, 1994.
 - [41] H. Ishibushi, H. Okada, H. Tanaka, "Fuzzy Neural Networks with Fuzzy Weights and Fuzzy Biases", *Proc. ICNN'93*, pp. 1650-1655, 1993.
 - [42] O. Cordon, F. Herrera, M. Lozano, "On the Combination of Fuzzy Logic and Evolutionary Computation: a Short Review and Bibliography, in: W. Pedrycz (Ed.)", *Fuzzy Evolutionary Computation*, Kluwer Academic Publishers, Dordrecht, pp. 33-56, 1997.
 - [43] A.B. Tickle, R. Andrews, M. Golea, J. Diederich, "The truth will come to light: Directions and challenges in extracting the knowledge embedded within trained artificial neural networks", *IEEE Trans. Neural Networks*, vol. 9, pp. 1057-1068, 1998.

Appendix A. Programs

A.1. Network_D.cs

```

namespace NSIM2
{
    public class Network
    {
        public static void Main()
        {
            System.Console.WriteLine("Loading network...");
            //Network nw= new Network("nsim.nw");
            //nw.InitializeWeights();
            Network nw= new Network(new int[] { 2, 3, 1 });
            nw.IO.Load("nsim.io");
            System.Console.WriteLine("Starting training...");
            nw.BP.Run(1000000, 10000);
            System.Console.Read();
        }

        int nlay;
        public int NLayers
        {
            get { return nlay; }
        }
        int[] units;
        public int[] Units
        {
            get { return units; }
        }
        double[][][] weight;
        double[][] netstate;
    }
}

```

```

protected double[] input;
protected double[] output;
TrainingData io;
BP_Training bp_training;
GA_Training ga_training;
bool trainable=false;
public TrainingData IO{ get{ return io; } }
public BP_Training BP{ get{ return bp_training; } }
public GA_Training GA{ get{ return ga_training; } }
public Network() : this(new int[2]{ 1, 1 })
{
}
public Network(int[] units)
{
    nlay= units.Length;
    if(nlay>4) nlay=4; else if(nlay<2) nlay=2;
    this.units= new int[nlay];
    int i;
    for(i=0; i<nlay; i++){
        int u= units[i];
        if(u<1) u=1; else if(u>999) u=999;
        this.units[i]=u;
    }
    weight= new double [nlay][][];
    int n;
    for(n=0; n<nlay-1; n++){
        weight[n]= new double[units[n+1]][];
        for(i=0; i<units[n+1]; i++){
            weight[n][i]= new double[units[n]+1];
        }
    }
    InitializeWeights();
    init_state();
}

```



```

        io= new TrainingData(this);
        bp_training= new BP_Training(this);
        ga_training= new GA_Training(this);
        trainable=false;
    }
    public Network(string nwfilename)
    {
        if(Load(nwfilename)==0)    Load();
        io = new TrainingData(this);
        bp_training= new BP_Training(this);
        ga_training= new GA_Training(this);
    }
    void init_state()
    {
        int i, n;
        netstate= new double[nlay][];
        for(n=0; n<nlay; n++) netstate[n]= new double[units[n]];
        for(i=0; i<units[0]; i++) netstate[0][i]=0;
        input=netstate[0];
        output=netstate[nlay-1];
        _Run();
    }
    void clear_netstate()
    {
        for(int n=0; n<nlay; n++) for(int i=0; i<units[n]; i++)
            netstate[n][i]=0;
    }
    public void InitializeWeights()
    {
        InitializeWeights(-0.05, 0.05);
    }
    public void InitializeWeights(double wmin, double wmax)
    {

```

```

int n,i,j;
System.Random rnd= new System.Random();
for(n=0; n<nlay-1; n++){
    for(i=0; i<units[n+1]; i++){
        for(j=0; j<units[n]+1; j++){
            weight[n][i][j]=
wmin+rnd.NextDouble()*(wmax-wmin);
        }
    }
}
protected int Load(string nwfilename)
{
    int n,i,j;
    sFILE f= new sFILE();
    if (f.OpenRead(nwfilename)==0)
    {
        // Error opening file
        return 0;
    }
    nlay=f.ReadInt();
    if (nlay>4||nlay<2)
    {
        return 0;
    }
    units= new int[nlay];
    for(n=0; n<nlay; n++){
        int un;
        un=f.ReadInt();
        units[n]= un>0 ? un : 1;
    }
    weight= new double[nlay][][];
    for(n=0; n<nlay-1; n++){

```

```

        weight[n]= new double[units[n+1]][];
        for(i=0; i<units[n+1]; i++){
            weight[n][i]= new double[units[n]+1];
        }
    }
    for(n=0; n<nlay-1; n++){
        for(i=0; i<units[n+1]; i++){
            for(j=0; j<units[n]+1; j++){
                weight[n][i][j]= f.ReadDouble();
            }
        }
    }
    f.Close();
    init_state();
    trainable=false;
    return 1;
}

void Load()
{
    nlay=2;
    units[0]=units[1]=1;
    weight= new double[nlay][][];
    weight[0]= new double[units[1]][];
    weight[0][0]= new double[2];
    init_state();
    trainable=false;
}

public int Save(string nwfilename)
{
    int n,i,j;
    sFILE f= new sFILE();
    if (f.OpenWrite(nwfilename)==0)
    {

```

```

        return 0;
    }
    f.Write(nlay+" "+units[0]+" "+units[1]);
    if(nlay==3) f.Write(" "+units[2]);
    else if(nlay==4) f.Write(" "+units[2]+" "+units[3]);
    f.Write("\n");
    for(n=0; n<nlay-1; n++){
        for(i=0; i<units[n+1]; i++){
            for(j=0; j<units[n]+1; j++){
                f.Write(weight[n][i][j]);
                f.Write("\n");
            }
        }
    }
    f.Close();
    return 1;
}

public void Run(double[] input, double[] output)
{
    this.input=input;
    this.output=output;
    _Run();
}

protected void _Run()
{
    int n,i,j;
    for(i=0; i<units[0]; i++) netstate[0][i]=input[i];
    for(n=0; n<nlay-1; n++){
        for(i=0; i<units[n+1]; i++){
            double state;
            state=weight[n][i][units[n]];
            for(j=0; j<units[n]; j++)
                state+=weight[n][i][j]*netstate[n][j];

```



```

        netstate[n+1][i]= Sigmoid(state);
    }
}
for(i=0; i<units[nlay-1]; i++) output[i]=netstate[nlay-1][i];
}
void _Run(double[] input)
{
    int n,i,j;
    for(i=0; i<units[0]; i++) netstate[0][i]= input[i];
    for(n=0; n<nlay-1; n++){
        for(i=0; i<units[n+1]; i++){
            double state;
            state=weight[n][i][units[n]];
            for(j=0; j<units[n]; j++)
                state+=weight[n][i][j]*netstate[n][j];
            netstate[n+1][i]= Sigmoid(state);
        }
    }
    for(i=0; i<units[nlay-1]; i++) output[i]=netstate[nlay-1][i];
}

public static double Sigmoid(double s)
{
    return s/(1+System.Math.Abs(s));
}

public static double DSigmoidS(double sigmoid)
{
    double f= 1-System.Math.Abs(sigmoid);
    return f*f;
}

/*
int series_length=5;
double[,] input_series;

```

```

double[,] output_series;
void _RunRecurrent(double[,] input_series, double[,] output_series)
{
    this.input_series=input_series;
    this.output_series=output_series;
    _RunRecurrent();
}
void _RunRecurrent()
{
    int t,n,i,j;
    clear_netstate();
    for(t=0; t<series_length; t++){
        for(i=0; i<units[0]; i++){
            netstate[0][i]=input_series[i,t];
        }
        for(n=0; n<nlay-1; n++){
            for(i=0; i<units[n+1]; i++){
                double state;
                state=weight[n][i][units[n]];
                for(j=0; j<units[n]; j++){
                    state+=weight[n][i][j]*netstate[n][j];
                }
                for(j=0; j<units[n+1]; j++){
                    state+=netstate[n+1][j];
                }
                netstate[n+1][i]= Sigmoid(state);
            }
        }
        for(i=0; i<units[nlay-1]; i++)
            output[i]=output_series[i,t]=netstate[nlay-1][i];
    }
}

```

```

public void AddTrainingData(int np)
{
    io= new TrainingData(this, np);
    trainable=true;
}

public void AddTrainingData(string iofilename)
{
    io= new TrainingData(this);
    io.Load(iofilename); // trainable=true;
}

public class TrainingData
{
    double[][] dInput;
    double[][] dOutput;
    public double[][] DInput{ get{ return dInput; } }
    public double[][] DOutput{ get{ return dOutput; } }
    int np, ni, no;
    public int NP{ get{ return np; } }
    public int NI{ get{ return ni; } }
    public int NO{ get{ return no; } }
    Network This;
    public TrainingData(Network nw)
    {
        This=nw;
        nw.io=this;
    }
    public TrainingData(Network nw, int np)
    {
        This = nw;
        int n;
        this.np= np>0 ? np : 1;
        this.ni=This.units[0];
        this.no=This.units[This.nlay-1];
    }
}

```

```

        dInput= new double[np][];
        for(n=0; n<np; n++) dInput[n]= new double[ni];
        dOutput= new double[np][];
        for(n=0; n<np; n++) dOutput[n]= new double[no];
        This.io=this;
        This.trainable=true;
    }
    public int Load(string iofilename)
    {
        int n, i;
        sFILE f= new sFILE();
        if (f.OpenRead(iofilename)==0)
        {
            return 0;
        }
        np=f.ReadInt();
        ni=f.ReadInt();
        no=f.ReadInt();
        if (ni!=This.units[0] || no!=This.units[This.nlay-1] ||
np<=0)

        {
            //Error in input or irrelevant data
            This.trainable=false;
            return 0;
        }
        dInput= new double[np][];
        for(n=0; n<np; n++) dInput[n]= new double[ni];
        dOutput= new double[np][];
        for(n=0; n<np; n++) dOutput[n]= new double[no];
        for(n=0; n<np; n++){
            for(i=0; i<ni; i++) dInput[n][i]=f.ReadDouble();
            for(i=0; i<no; i++) dOutput[n][i]=f.ReadDouble();
        }
    }

```



```

        This.io=this;
        This.trainable=true;
        return 1;
    }
    public double GetMSE()
    {
        double error=0.0;
        int p, k;
        for(p=0; p<This.io.NP; p++){
            This._Run(This.io.DInput[p]);
            for(k=0; k<This.units[This.nlay-1]; k++){
                double e=dOutput[p][k]-This.output[k];
                error+=e*e;
            }
        }
        return error/This.io.NP;
    }
    public double GetRMSE()
    {
        return System.Math.Sqrt(GetMSE());
    }
    int _Load_R(string iofilename)
    {
        if(_Load(iofilename)==0) return 0;
        if(np<This.series_length) np=This.series_length;
        This.input_series= new double[ni,This.series_length];
        This.output_series= new double[no,This.series_length];
        return 1;
    }
    int Load_R(string iofilename)
    {
        return _Load_R(iofilename);
    }

```

```

    }
    public class BP_Training
    {
        double rate=0.1;
        //double momentum=0.9;
        Network This;
        public double Rate{ get{ return rate; } set{ rate= value>0 ? value
: 0.1; } }

        public void Initialize()
        {
            This.InitializeWeights();
        }
        double[][] error;
        public BP_Training(Network nw)
        {
            This=nw;
            int l;
            error= new double[This.nlay-1][];
            for(l=0; l<This.nlay-1; l++) error[l]= new
double[This.units[l+1]];
        }
        public void Run(int steps)
        {
            Run(steps, 1);
        }
        public void Run(int steps, int showevery)
        {
            if(steps<=0) return;
            if(showevery>steps) showevery=steps;
            int s;
            if(This.trainable){
                System.Console.WriteLine("BackPropagation
Training Algorithm\n\n");

```

```

        for(s=0; s<steps; s++){
            _Step();
            if(s>0 && s%showevery==0){
                //for(i=0;
i<(s+""+TotError).Length+24; i++) System.Console.Write("\b");
                System.Console.Write("Step: "+s+"
RMSE: "+This.io.GetRMSE()+"\n");
            }
        }
        System.Console.WriteLine("\n\nTraining
completes:\n");
        System.Console.Write("Step: "+s+" Final RMSE:
"+This.io.GetRMSE()+"\n\n");
    }
}
public void Run()
{
    Run(1);
}
protected internal void _Step()
{
    int p, l, i, j, k;
    double d, dw;
    //TotError=MaxError=0.0;
    for(p=0; p<This.io.NP; p++){
        This._Run(This.io.DInput[p]);
        for(i=0; i<This.units[This.nlay-1]; i++){
            double e= This.io.DOutput[p][i]-
This.output[i];

            //TotError+=e*e;
            //if(MaxError<System.Math.Abs(e))
MaxError=System.Math.Abs(e);

            d= e*DSigmoidS(This.output[i]);

```

```

for(j=0; j<This.units[This.nlay-2]; j++){
    dw=
rate*d*This.netstate[This.nlay-2][j];

    //_weight(nlay-1, i, j)+=dw;
    This.weight[This.nlay-
2][i][j]+=dw;
}
This.weight[This.nlay-2][i][j]+=rate*d; //
adjust threshold - the last link

    //_error(nlay-1, i)=d;
    error[This.nlay-2][i]=d;
}
for(l=This.nlay-2; l>0; l--){
    for(i=0; i<This.units[l]; i++){
        d=0;
        for(k=0; k<This.units[l+1]; k++){
            //d+=_error(l+1,
k)*_weight(l+1, k, i);

            d+=error[l+1][k]*This.weight[l+1][k][i];
        }
        d*= DSigmoidS(This.netstate[l][i]);
        for(j=0; j<This.units[l-1]; j++){
            dw=rate*d*This.netstate[l-
1][j];

            //_weight(l, i, j)+=dw;
            This.weight[l-1][i][j]+=dw;
        }
        This.weight[l-1][i][j]+=rate*d;
        //_error(l, i)=d;
        error[l-1][i]=d;
    }
}

```



```

    }
}

public class GA_Training
{
    Network This;

    public GA_Training(Network nw)
    {
        This=nw;
    }
}
}

```

A.2 Controller.cs

```

class Controller: NSIM2.Network
{
    public static void Main()
    {
        Controller controller= new Controller(
            new Plant(new Plant.Model(ThisPlantModel)),
            new ReferencePlant(
                new ReferencePlant.Model(ThisReferenceModel),
                new ReferencePlant.ReferenceInput(ThisReferenceInput)
            ),
            "identifier.nw"
        );

        System.Console.WriteLine("Testing Plant outputs...");
        controller.ShowPlantOutput(100);
        System.Console.WriteLine("\nPlant Identification process started...");
        controller.DoIdentification(2000, 100000);
    }
}

```

```

    controller.Run(2000, 0.02);
    System.Console.Read();
}

static double ThisPlantModel(Plant.State y, double u)
{
    return 0.35*(y[0]*y[1]*(y[0]+2.5)/(1+y[0]*y[0]+y[1]*y[1])+u);
}

static double ThisReferenceInput(int t)
{
    if(t<500) return System.Math.Sin(System.Math.PI*t/25);
    else if(t<1000) return 1;
    else if(t<1500) return -1;
    else return
        0.3*System.Math.Sin(System.Math.PI*t/25)+
        0.4*System.Math.Sin(System.Math.PI*t/32)+
        0.3*System.Math.Sin(System.Math.PI*t/10);
}

static double ThisReferenceModel(Plant.State ym, double r)
{
    return 0.6*ym[0]+0.2*ym[1]+0.1*r;
}

Plant ThisPlant;
ReferencePlant ThisReference;
Controller(Plant p): base( new int[] { 4, 20, 1 } )
{
    ThisPlant=p;
}

public Controller(Plant p, ReferencePlant r): this(p)
{
    ThisReference=r;
}

public Controller(Plant p, ReferencePlant r, string filename): this(p,r)
{

```

```

        this.Load(filename);
    }
    public void DoIdentification(int ndata, int nlessons)
    {
        this.AddTrainingData(ndata);
        int t;
        for(t=0; t<ndata; t++){
            double refinp= ThisReference.Input(t);
            ThisPlant.Feed(refinp);
            this.IO.DInput[t][0]= ThisPlant.Output;
            this.IO.DInput[t][1]= ThisPlant.OutputZ[1];
            this.IO.DInput[t][2]= ThisPlant.OutputZ[2];
            this.IO.DInput[t][3]= ThisPlant.OutputZ[3];
            this.IO.DOutput[t][0]= refinp;
        }
        this.BP.Run(nlessons, nlessons/100);
        this.Save("Identifier.nw");
    }

    public void Run(int ndata)
    {
        int t;
        for(t=0; t<ndata; t++){
            ThisReference.Feed(t);
            this.input[0]= ThisReference.Output;
            this.input[1]= ThisPlant.Output;
            this.input[2]= ThisPlant.OutputZ[1];
            this.input[3]= ThisPlant.OutputZ[2];
            this._Run();
            ThisPlant.Feed(this.output[0]);
            System.Console.WriteLine(ThisPlant.Output+"\t\t"+ThisReference.Output);
        }
    }

    public void Run(int ndata, double maxerr)

```

```

{
    int t;
    for(t=0; t<ndata; t++){
        ThisReference.Feed(t);
        this.input[0]= ThisReference.Output;
        this.input[1]= ThisPlant.Output;
        this.input[2]= ThisPlant.OutputZ[1];
        this.input[3]= ThisPlant.OutputZ[2];
        this._Run();
        double control_input=this.output[0];
        ThisPlant.Feed(control_input);
        System.Console.Write(ThisPlant.Output+"\t\t"+ThisReference.Output);
        double err=System.Math.Abs(ThisPlant.Output-ThisReference.Output);
        if(err>maxerr){
            System.Console.Write("  ---Training...");
            this.AddTrainingData(1);
            this.IO.DInput[0][0]= ThisReference.Output;
            this.IO.DInput[0][1]= ThisReference.OutputZ[1];
            this.IO.DInput[0][2]= ThisReference.OutputZ[2];
            this.IO.DInput[0][3]= ThisReference.OutputZ[3];
            this.IO.DOutput[0][0]= ThisReference.Input(t); // not possible!
            for(int i=0; i<10000; i++) this.BP._Step();
        }
        System.Console.WriteLine();
    }
}

public void ShowPlantOutput(int nsteps)
{
    int t;
    for(t=0; t<nsteps; t++){
        ThisPlant.Feed(ThisReference.Input(t));
        System.Console.WriteLine(ThisPlant.Output);
    }
}

```



```

    }
}
public class Plant
{
    const int MaxOrder= 5;
    Model ThisModel;
    State ThisState;
    public State OutputZ{ get { return ThisState; } }
    public delegate double Model(Plant.State state, double input);
    public Plant(Model model)
    {
        ThisModel= model;
        ThisState= new State();
    }
    public double Output{ get{ return ThisState.Now; } }
    public class State
    {
        internal double[] item= new double[MaxOrder];

        public double Now{ get{ return item[0]; } }
        public double Prev{ get{ return item[1]; } }
        public double PrevPrev{ get{ return item[2]; } }
        public double this[int i]
        {
            get{
                return System.Math.Abs(i)<MaxOrder ? item[System.Math.Abs(i)] : 0;
            }
        }
    }
}
    public void Feed(double input)
    {
        double next_state= ThisModel(ThisState, input);
        ThisState.item[4]= ThisState.item[3];
    }
}

```

```

        ThisState.item[3]= ThisState.item[2];
        ThisState.item[2]= ThisState.item[1];
        ThisState.item[1]= ThisState.item[0];
        ThisState.item[0]= next_state;
    }
}

public class ReferencePlant: Plant
{
    public ReferencePlant(Model model, ReferenceInput refinp): base(model)
    {
        Input= refinp;
    }
    public ReferenceInput Input;
    public delegate double ReferenceInput(int t);
    new void Feed(double input){}
    public void Feed(int t)
    {
        base.Feed(Input(t));
    }
}

```

A.3 Charging_control.cs

```

public class ChargingController
{
    public static void Main()
    {
        ChargerModel Charger= new ChargerModel();
        double c;
        int t;
        int k=0;
        for(c=-0.75; c<0.751; c+=0.05){
            for(t=0; t<1000; t+=10){

```

```

        System.Console.Write(Charger.VoltageNN(c, t)+"\t");
        System.Console.Write(Charger.VoltageNN(c, t-1)+"\t");
        System.Console.WriteLine(c);
        k++;
    }
}

System.Console.WriteLine(k);
System.Console.Read();
}
}

public class ChargerModel
{
    NSIM2.Network nw_vol;
    NSIM2.Network nw_tem;

    double[] input;
    double[] output;

    public ChargerModel()
    {
        nw_vol= new NSIM2.Network("charger-vol.nw");
        nw_tem= new NSIM2.Network("charger-tem.nw");
        input= new double[1];
        output= new double[1];
    }

    public double VoltageNN(double currency, int tau)
    {
        double x= -0.75+1.5*tau/1050;
        if(x<-1) x=-1.0; else if(x>1) x=1.0;
        input[0]=x;
        nw_vol.Run(input, output);
        return currency*output[0]/6;
    }

    public double TemperatureNN(double currency, int tau)

```

```

{
    double x= -0.75+1.5*tau/1050;
    if(x<-1) x=-1.0; else if(x>1) x=1.0;
    input[0]=x;
    nw_tem.Run(input, output);
    return currency*output[0]/6;
}

public double Voltage(double currency, int tau)
{
    return (1.362666667+(1.493333333-1.362666667)*(0.75+VoltageNN(currency,
tau))*2/3)*8;
}

public double Temperature(double currency, int tau)
{
    return (21.57333333+(27.306666673-
21.57333333)*(0.75+TemperatureNN(currency, tau))*2/3)*8;
}
}

```

A.4 FR_ChargingControl.cs

```

class Controller
{
    public static void Main()
    {
        Controller controller= new Controller();
        //
        // show model
        controller.Run_Charging();
        //
        System.Console.Read();
        //
    }
}

```



```

}
static Battery ThisBattery= new Battery();
static double vlt= ThisBattery.Voltage0;
static double tem= ThisBattery.Temperature0;
static double dvlt=0;
static double dtem=0;
void Run_Charging()
{
    int i;
    ThisBattery.Discharge();
    //
    /*
    for(i=0; i<600; i++){
        //System.Console.WriteLine("Vlt:\t"+vlt+"\t"+"Tem\t"+tem);
        //System.Console.WriteLine(vlt);
        System.Console.WriteLine(tem);
        //if(vlt>1.6) goto end;
        Run_Charging(6);
    }
    for(i=0; i<1800; i++){
        //System.Console.WriteLine("Vlt:\t"+vlt+"\t"+"Tem\t"+tem);
        //System.Console.WriteLine(vlt);
        System.Console.WriteLine(tem);
        //if(vlt>1.6) break;
        Run_Charging(2);
    }
    end: ;
    Control_Charging();
}
void Run_Charging(double cur)
{
    ThisBattery.Current=cur;
    dvlt= ThisBattery.VoltageChange;

```

```

        vlt+= dvlt;
        dtem= ThisBattery.TemperatureChange;
        tem+= dtem;
        ThisBattery.Consume();
    }
    double Min(double x1, double x2)
    {
        return System.Math.Min(x1, x2);
    }
    double Min(double x1, double x2, double x3)
    {
        return System.Math.Min(System.Math.Min(x1, x2), x3);
    }
    double Max(double x1, double x2)
    {
        return System.Math.Max(x1, x2);
    }
    double Max(double x1, double x2, double x3)
    {
        return System.Math.Max(System.Math.Max(x1, x2), x3);
    }
    double mf_Tem_is_LOW(double x)
    {
        double c=24;
        double d=26;
        return Max(Min(1, (d-x)/(d-c)), 0);
    }
    double mf_Tem_is_MED(double x)
    {
        double a=25;
        double b=26;
        double c=27;
        double d=28;

```

```

        return Max(Min((x-a)/(b-a), 1, (d-x)/(d-c)), 0);
    }

    double mf_Tem_is_HIGH(double x)
    {
        double a=27.5;
        double b=28;
        return Max(Min((x-a)/(b-a), 1), 0);
    }

    double mf_dTem_is_LOW(double x)
    {
        double c=7.5*0.002;
        double d=17*0.002;
        return Max(Min(1, (d-x)/(d-c)), 0);
    }

    double mf_dTem_is_MED(double x)
    {
        double a=7.5*0.002;
        double b=14*0.002;
        double c=19*0.002;
        double d=27*0.002;
        return Max(Min((x-a)/(b-a), 1, (d-x)/(d-c)), 0);
    }

    double mf_dTem_is_HIGH(double x)
    {
        double a=17*0.002;
        double b=27*0.002;
        return Max(Min((x-a)/(b-a), 1), 0);
    }

    double mf_Vol_is_LOW(double x)
    {
        double c=0.4;
        double d=0.6;
        return Max(Min(1, (d-x)/(d-c)), 0);
    }

```

```

}
double mf_Vol_is_MED(double x)
{
    double a=0.4;
    double b=0.6;
    double c=1.58;
    double d=1.61;
    return Max(Min((x-a)/(b-a), 1, (d-x)/(d-c)), 0);
}

```

```

double mf_Vol_is_HIGH(double x)
{
    double a=1.6;
    double b=1.65;
    return Max(Min((x-a)/(b-a), 1), 0);
}

```

```

double mf_dVol_is_LOW(double x)
{
    double c=-1.5*0.0001;
    double d=2.5*0.0001;
    return Max(Min(1, (d-x)/(d-c)), 0);
}

```

```

double mf_dVol_is_MED(double x)
{
    double a=-0.5*0.0001;
    double b=1.5*0.0001;
    double c=3.5*0.0001;
    double d=5.5*0.0001;
    return Max(Min((x-a)/(b-a), 1, (d-x)/(d-c)), 0);
}

```

```

double mf_dVol_is_HIGH(double x)
{
    double a=2.5*0.0001;
    double b=5.5*0.0001;

```



```

        return Max(Min((x-a)/(b-a), 1), 0);
    }

    void Control_Charging()
    {
        NSIM2.Network nw= new NSIM2.Network("rules.nw");
        double[] input= new double[12];
        double[] output= new double[1];
        int i;
        for(i=0; i<2400; i++){
            dvlt= ThisBattery.VoltageChange;
            vlt+= dvlt;
            dtem= ThisBattery.TemperatureChange;
            tem+= dtem;
            input[0]= mf_Tem_is_LOW(tem);
            input[1]= mf_Tem_is_MED(tem);
            input[2]= mf_Tem_is_HIGH(tem);
            input[3]= mf_dTem_is_LOW(dtem);
            input[4]= mf_dTem_is_MED(dtem);
            input[5]= mf_dTem_is_HIGH(dtem);
            input[6]= mf_Vol_is_LOW(vlt);
            input[7]= mf_Vol_is_MED(vlt);
            input[8]= mf_Vol_is_HIGH(vlt);
            input[9]= mf_dVol_is_LOW(dvlt);
            input[10]= mf_dVol_is_MED(dvlt);
            input[11]= mf_dVol_is_HIGH(dvlt);
            nw.Run(input, output);
            ThisBattery.Current=((int)(output[0]/2+0.5))*2;
            //System.Console.WriteLine(ThisBattery.Current);
            //System.Console.WriteLine(vlt);
            //System.Console.WriteLine(tem);
            ThisBattery.Consume();
        }
    }
}

```

```
}
```

A.5 TrainRules.cs

```
public class TrainRules
{
    public static void Main()
    {
        NSIM2.Network nw= new NSIM2.Network(new int[] { 12, 6, 1 } );
        nw.IO.Load("rules.io");
        System.Console.WriteLine("Rate="+nw.BP.Rate);
        nw.BP.Run(10000, 1000);
        nw.Save("rules.nw");
        System.Console.Read();
    }
}
```

A.6 Genetic_D.cs

```
using System;
/*
class test
{
    class FunMinimize: Evolution.Problem
    {
        internal FunMinimize()
        {
            parameter_number=2;
            av0=0;
            av1=1;
            parameters= new double[2]{ 0.5, 0.5 };
        }
    }
}
```

```

    }
    double fun(double x, double y)
    {
        return x*x+y*y+x*y+1/x+1/y;
    }

    override public double Fitness
    {
        get{
            return fun(parameters[0], parameters[1]);
        }
    }
}

public static void Main()
{
    Console.WriteLine("Optimization started...");
    Evolution.Problem p= new FunMinimize();
    Evolution e= new Evolution(p);
    e.Run(500, 50, "minimize.log");
    Console.WriteLine("Optimization stopped...");
    Console.Read();
}

}

public class Evolution
{
    public class Problem
    {
        protected internal int parameter_number=1;
        public int ParameterNumber{ get{ return parameter_number; } }
        protected internal double av0=0;
        //public double Av0{ get { return av0; } }
        protected internal double av1=10;
        //public double Av1{ get { return av1; } }
        protected internal double[] parameters= new double[1];
    }
}

```

```

    public double[] Parameters{ get{ return parameters; } }
    public virtual double Fitness{ get{ return 100; } }
    public virtual int Save(string problemfilename)
    {
        sFILE f= new sFILE();
        f.OpenWrite(problemfilename);
        for(int i=0; i<parameter_number; i++)
            f.Write("Parameter["+i+"]: "+parameters[i]+"\\n");
        f.Write("Fitness: "+Fitness+"\\n");
        f.Write("\\n");
        f.Close();
        return 1;
    }
}

Problem This= new Problem
uint av_max;
double av_range;

void initialize_bits()
{
    //av_bits=32;
    av_max=0xffffffff;
    av_range=(double)av_max+1.0;
}

int pop_size=100;
int num_best=10;
public void SetPopulationParams(int psize, int nbest)
{
    if(psize>0) pop_size=psize;
    if(nbest<pop_size) num_best=nbest;
    else{ num_best=pop_size/10; if(num_best==0) num_best=1; }
}

int ngens;
int[] weights;

```



```

public class Genome
{
    public System.Collections.BitArray bits;
    public double Fitness;
    public readonly int Length;
    public Genome(int[] ws)
    {
        bits= new System.Collections.BitArray(ws);
        Length=ws.Length*32;
        Fitness=double.MaxValue; // the worst
    }
    public Genome(int n)
    {
        Length=n*32;
        bits= new System.Collections.BitArray(Length);
        Fitness=double.MaxValue; // the worst
    }
    public Genome(Genome g)
    {
        Length=g.Length;
        bits= new System.Collections.BitArray(g.bits);
        Fitness=g.Fitness;
    }
    public uint Get32(int i)
    {
        int k0=32*i;
        int k1=k0+32;
        int k;
        uint w=0;
        for(k=k1-1; k>=k0; k--){ w<<=1; if(bits[k]) w|=1; }
        return w;
    }
    public bool Get(int i)

```

```

    {
        return bits.Get(i);
    }
    public int GetBit(int i)
    {
        return bits.Get(i) ? 1 : 0;
    }
    public void Set(int i, bool v)
    {
        bits.Set(i, v);
    }
    public void SetBit(int i, int v)
    {
        bits.Set(i, v==1 ? true : false);
    }
    public void InvertBit(int i)
    {
        bits.Set(i, !bits.Get(i));
    }
}

Genome[] genomes;
System.Random rnd= new System.Random(137);
static double av0;
static double av1;
static int parameter_number;
public Evolution(Problem p)
{
    initialize_bits();
    This=p;
    ngens= This.parameter_number;
    av0= This.av0;
    av1= This.av1;
    parameter_number= This.parameter_number;
}

```

```

        genomes= new Genome[pop_size];
        genomes[0]=_GetGenotype();
    }
    public Evolution(): this(new Problem())
    {
    }
    Genome _GetGenotype()
    {
        int l;
        weights= new int[ngens];
        for(l=0; l<parameter_number; l++){
            weights[l]=(int)_Encode(This.parameters[l]);
            /*
            System.Console.WriteLine(This.parameters[l]);
//System.Console.WriteLine(GenToString((uint)weights[l]));
            System.Console.ReadLine();
            */
        }
        return new Genome(weights);
    }
    void _GetPhenotype(Genome g)
    {
        int l;
        for(l=0; l<parameter_number; l++)
        {
            //This.Parameters[l]=_Decode((uint)weights[l]);
            This.parameters[l]=_Decode(g.Get32(l));
            /*
            System.Console.WriteLine(This.parameters[l]);
            System.Console.ReadLine();
            */
        }
    }
}

```

```

void con(string s){ System.Console.WriteLine(s); }
void _con(string s){ System.Console.Write(s); }
void con(uint gen)
{
    int i;
    string s="", s1="";
    for(i=0; i<32; i++){ s+=gen%2; gen/=2; }
    for(i=0; i<32; i++) s1+=s[31-i];
    con(s1);
}

public string GenToString(uint gen)
{
    int i;
    string s="", s1="";
    for(i=0; i<32; i++){ s+=gen%2; gen/=2; }
    for(i=0; i<32; i++){ s1+=s[31-i]; }
    return s1;
}

public string GenomeToString(int n)
{
    int i;
    string s="";
    for(i=0; i<genomes[n].Length; i++){
        if(i>0&& i%32==0) s+="\n";
        s+= genomes[n].GetBit(i);
    }
    return s;
}

public uint _Encode(double w)
{
    if(w>=av1) return 0xffffffff;
    else if(w<=av0) return 0;
}

```



```

        double v= (w-av0)/(av1-av0)*av_range+0.5;
        if(v>av_range) return 0xffffffff;
        else return (uint)v;
    }
    public double _Decode(uint c)
    {
        return c*(av1-av0)/av_range+av0;
    }
    Genome[] offsprings;
    /*
    double xov_prob=0.25;
    double mut_prob=0.05;
    */
    double xov_prob=0.50;
    double mut_prob=0.05;
    double _DetermineFitnessValue(Genome g)
    {
        _GetPhenotype(g);
        return g.Fitness=This.Fitness;
    }//
    int genome_length;
    int number_offsprings;
    void _RecallBestAvailableGenome()
    {
        genomes[0]=_GetGenotype();
    }
    void _GetRandomPopulation()
    {
        number_offsprings=pop_size*(pop_size-1);
        offsprings= new Genome[number_offsprings];
        genome_length=genomes[0].Length;
        int i;
        int k;

```

```

    k=1;
    for(; k<pop_size; k++){
        for(i=0; i<ngens; i++){
            weights[i]=rnd.Next()+rnd.Next();
        }
        genomes[k]= new Genome(weights);
    }
}

void _GeneratePopulation()
{
    int i;
    int k, l, r;
    r=0;
    for(k=0; k<pop_size; k++) for(l=0; l<k; l++){
        offsprings[r]= new Genome(genomes[k]);
        for(i=0; i<genome_length; i++){
            if(rnd.NextDouble()<xov_prob)
                offsprings[r].Set(i, genomes[l].Get(i));
        }
        r++;
        offsprings[r]= new Genome(genomes[l]);
        for(i=0; i<genome_length; i++){
            // inherit bit <i> in genome <l> from genome <k>
            // with probability <xov_prob>
            if(rnd.NextDouble()<xov_prob)
                offsprings[r].Set(i, genomes[k].Get(i));
        }
        r++;
    }
    for(k=0; k<r; k++){
        for(i=0; i<genome_length; i++){
            <mut_prob>

```

```

        if(rnd.NextDouble()<mut_prob)
    offsprings[k].InvertBit(i);
    }
}
double _SelectPopulation()
{
    int k, l;
    double[] sort_keys= new double[number_offsprings+num_best];
    Genome[] sort_items= new
Genome[number_offsprings+num_best];
    k=0;
    for(l=0; l<num_best; l++){
        sort_items[k]=genomes[l];
        sort_keys[k]=_DetermineFitnessValue(genomes[l]);
        k++;
    }
    for(l=0; l<number_offsprings; l++){
        sort_items[k]=offsprings[l];
        sort_keys[k]=_DetermineFitnessValue(offsprings[l]);
        k++;
    }
    System.Array.Sort(sort_keys, sort_items);
    for(k=0; k<pop_size; k++){
        genomes[k]=sort_items[k];
    }
    return sort_keys[0]; // best fitness value
}
int _Save(string problemfilename)
{
    _GetPhenotype(genomes[0]);
    return This.Save(problemfilename);
}

```

```

public double Run(int nregen, int saveevery, string problemfilename)
{
    con("\n");
    con("Genetic Algorithm Based Optimization\n\n");
    con("Problem name: "+This+"\n");
    con("GA parameters:");
    con("_____ \n");
    con("Size of population:          "+pop_size);
    con("Number of best parent genomes to save: "+num_best);
    con("Probability of crossover:      "+xov_prob);
    con("Probability of mutation:       "+mut_prob);
    con("_____ \n");
    con("");
    con("Present fitness value:
"+_DetermineFitnessValue(_GetGenotype()));
    con("");
    int n;
    double oldfit=0.0;
    double bestfit=100.0;
    int gener=0;
    if(nregen<=0) return bestfit;
    if(saveevery<=0) saveevery=nregen;
    _RecallBestAvailableGenome();
    _GetRandomPopulation();
    for(n=0; n<nregen/saveevery; n++){
        for(int g=0; g<saveevery; g++){
            _GeneratePopulation();
            bestfit=_SelectPopulation();
            //
            for(int i=0; i<(gener+ "").Length+13; i++)
                _con("\b");
            _con("Generations: "+(++gener));
            if(bestfit!=oldfit){

```



```

        con("\nBest fitness value:
"+(oldfit=bestfit)+"\n");
    }
}
_Save(problemfilename);
}
for(int g=0; g<nregen%saveevery; g++){
    _GeneratePopulation();
    bestfit=_SelectPopulation();
    //
    for(int i=0; i<(gener+ "").Length+13; i++) _con("\b");
    _con("Generations: "+(++gener));
    if(bestfit!=oldfit){
        con("\nBest fitness value: "+(oldfit=bestfit)+"\n");
    }
    //
} //g
_Save(problemfilename);
return bestfit;
} //Run method
public bool Run(int maxniter, double maxerr)
{
    con("\n");
    con("GA Optimization --> Problem name: "+This+"\n");
    con("Present fitness value:
"+_DetermineFitnessValue(_GetGenotype()));
    int n;
    double oldfit=0.0;
    double bestfit=100.0;
    int gener=0;
    if(maxniter<=0) return false;
    _RecallBestAvailableGenome();
    GetRandomPopulation();

```

```

        for(n=0; n<maxniter; n++){
            _GeneratePopulation();
            bestfit=_SelectPopulation();
            if(bestfit!=oldfit){
                _con("Generations: "+(++gener));
                con("\tBest fitness value: "+(oldfit=bestfit));
            }
            if(bestfit<=maxerr) break;
        }//n
        // _con("Generations: "+(++gener));
        //con("\tBest fitness value: "+(oldfit=bestfit)+"\n");
        return bestfit<=maxerr;
    }
}

```

A.7 Battery.cs

```

public class Battery
{
    static double scale_input(double x, double c)
    {
        return -0.75 + 1.5 * x * (c / 6) / 1050;
    }
    static double V_scale_output(double x)
    {
        return 1.36 + (1.60 - 1.36) * 2 / 3 * (0.75 + x);
    }
    static double[,] V_W1= new double[3, 2]{
        { -8.128643533, -5.662001088 },
        { 8.697642486, -5.176190576 },
        { 9.392557863, -6.166325547 }
    };
}

```

```

static double[] V_W2= new double[4]{
    -2.41976912, 7.418189749, -7.849240231, -2.7049977
};
double[] V_state1= new double[3];
double V_state2;
double V_Output(double cur, int dt)
{
    int i;
    double input= scale_input(dt, cur);
    for(i=0; i<3; i++){
        V_state1[i]= input*V_W1[i, 0]+V_W1[i, 1];
        V_state1[i]= V_state1[i]/(1+System.Math.Abs(V_state1[i]));
    }
    V_state2=
V_state1[0]*V_W2[0]+V_state1[1]*V_W2[1]+V_state1[2]*V_W2[2]+V_W2[3];
    return V_scale_output(V_state2/(1+System.Math.Abs(V_state2)));
}
int time=0;
public void Discharge()
{
    time=0;
    current=6;
    voltage= 0;
    Voltage0= V_Output(current, time);
    Temperature0= T_Output(current, time);
    int i;
    for(i=0; i<200; i++) voltage_history[i]=0;
    for(i=0; i<200; i++) temperature_history[i]=0;
    time++;
}
public double Voltage0;
public double Current{ get{ return current; } set{ current=value; } }
double current=6;

```

```

public double VoltageChange{ get{ return GetVoltage(); } }
double voltage;
double[] voltage_history= new double[200];
double GetVoltage()
{
    int i;
    for(i=0; i<=198; i++) voltage_history[199-i]=voltage_history[198-i];
    voltage_history[0]= V_Output(current, time)-V_Output(current, time-1);
    voltage=0;
    for(i=0; i<200; i++) voltage+= voltage_history[i]*0.383/(2*i+1);
    return voltage;
}
static double T_scale_output(double x)
{
    return 21.573 + (30.307 - 21.573) * 2 / 3 * (0.75 + x);
}
static double[,] T_W1= new double[3, 2]{
    { 11.18278511, -8.128164021 },
    { 7.573209499, -2.448374555 },
    { 13.36306043, -8.307475498 }
};
static double[] T_W2= new double[4]{
    6.94953707, 0.831031877, -1.005212074, 3.448737096
};
double[] T_state1= new double[3];
double T_state2;
public double T_Output(double cur, int dt)
{
    int i;
    double input= scale_input(dt, cur);
    for(i=0; i<3; i++){
        T_state1[i]= input*T_W1[i, 0]+T_W1[i, 1];
        T_state1[i]= T_state1[i]/(1+System.Math.Abs(T_state1[i]));
    }
}

```



```

    }
    T_state2=
T_state1[0]*T_W2[0]+T_state1[1]*T_W2[1]+T_state1[2]*T_W2[2]+T_W2[3];
    return T_scale_output(T_state2/(1+System.Math.Abs(T_state2)));
}
public double Temperature0;
public double TemperatureChange{ get{ return GetTemperature(); } }
double temperature;
double[] temperature_history= new double[200];
double GetTemperature()
{
    int i;
    for(i=0; i<=198; i++) temperature_history[199-
i]=temperature_history[198-i];
    temperature_history[0]= T_Output(current, time)-T_Output(current,
time-1);
    temperature=0;
    for(i=0; i<200; i++) temperature+=
temperature_history[i]*0.383/(2*i+1);
    return temperature;
}
public Battery()
{
    Discharge();
}
public void Consume()
{
    time++;
}
}

```

A.8 NSIM-tem.xls file macro written with Visual Basic

```

Dim W1() As Double
Dim W2() As Double
Dim NI As Integer, NH As Integer, NO As Integer
Dim WeightsLoaded As Boolean
'WeightsLoaded = False
Dim StateAllocated As Boolean
'StateAllocated = False
Dim State1() As Double
Dim State2() As Double
Function scale_input(x As Double, i As Double)
    scale_input = -0.75 + 1.5 * x * (i / 6) / 1050
End Function
Function scale_output(x As Double)
    scale_output = 21.57333333 + (27.306666673 - 21.57333333) * 2 / 3 * (0.75 + x)
End Function
Sub Load()
    Dim i As Integer, j As Integer, k As Integer
    NI = Range("NInputs").Cells.Value
    NH = Range("NHidden").Cells.Value
    NO = Range("NOutputs").Cells.Value
    Dim Weights As Range
    Set Weights = Range("Weights")
    ReDim W1(NH, NI + 1) As Double
    ReDim W2(NO, NH + 1) As Double
    k = 1
    For i = 1 To NH
        For j = 1 To NI + 1
            W1(i, j) = Weights.Cells(k, 1).Value
            k = k + 1
        Next j
    Next i
    Next i

```

```

For i = 1 To NO
    For j = 1 To NH + 1
        W2(i, j) = Weights.Cells(k, 1).Value
        k = k + 1
    Next j
Next i
WeightsLoaded = True
End Sub

Sub StateAllocate()
    ReDim State1(NH)
    ReDim State2(NO)
    StateAllocated = True
End Sub

Public Function GetOutput(n As Integer, Inp As Range)
    Dim i As Integer, j As Integer
    If (Not WeightsLoaded) Then Call Load
    If (Not StateAllocated) Then Call StateAllocate
    For i = 1 To NH
        State1(i) = 0
        For j = 1 To NI
            State1(i) = State1(i) + W1(i, j) * Inp.Cells(1, j).Value
        Next j
        State1(i) = State1(i) + W1(i, NI + 1)
        State1(i) = State1(i) / (1 + Abs(State1(i)))
    Next i
    For i = 1 To NO
        State2(i) = 0
        For j = 1 To NH
            State2(i) = State2(i) + W2(i, j) * State1(j)
        Next j
        State2(i) = State2(i) + W2(i, NH + 1)
        State2(i) = State2(i) / (1 + Abs(State2(i)))
    Next i

```

```

    GetOutput = State2(n)
End Function
Function frange(i As Integer, r As Range)
    frange = r.Cells(1, i).Value
End Function
Sub test()
    Dim r As Range
    'If (Not WeightsLoaded) Then Call Load
    'If (Not StateAllocated) Then Call StateAllocate
    Set r = Sheets("Simulation").Range("B4:C4")
    Sheets("Simulation").Cells(4, 6) = GetOutput(1, r)
    Set r = Sheets("Simulation").Range("B5:C5")
    Sheets("Simulation").Cells(5, 6) = GetOutput(1, r)
    Set r = Sheets("Simulation").Range("B6:C6")
    Sheets("Simulation").Cells(6, 6) = GetOutput(1, r)
    Set r = Sheets("Simulation").Range("B7:C7")
    Sheets("Simulation").Cells(7, 6) = GetOutput(1, r)
End Sub

```

A.9 NSIM-vol.xls file macro written with Visual Basic

```

Dim W1() As Double
Dim W2() As Double
Dim NI As Integer, NH As Integer, NO As Integer
Dim WeightsLoaded As Boolean
'WeightsLoaded = False
Dim StateAllocated As Boolean
'StateAllocated = False
Dim State1() As Double
Dim State2() As Double
Function scale_input(x As Double, i As Double)
    scale_input = -0.75 + 1.5 * x * (i / 6) / 1050

```


End Function

Function scale_output(x As Double)

scale_output = $1.362666667 + (1.493333333 - 1.362666667) * 2 / 3 * (0.75 + x)$

End Function

Sub Load()

Dim i As Integer, j As Integer, k As Integer

NI = Range("NInputs").Cells.Value

NH = Range("NHidden").Cells.Value

NO = Range("NOutputs").Cells.Value

Dim Weights As Range

Set Weights = Range("Weights")

ReDim W1(NH, NI + 1) As Double

ReDim W2(NO, NH + 1) As Double

k = 1

For i = 1 To NH

For j = 1 To NI + 1

W1(i, j) = Weights.Cells(k, 1).Value

k = k + 1

Next j

Next i

For i = 1 To NO

For j = 1 To NH + 1

W2(i, j) = Weights.Cells(k, 1).Value

k = k + 1

Next j

Next i

WeightsLoaded = True

End Sub

Sub StateAllocate()

ReDim State1(NH)

ReDim State2(NO)

StateAllocated = True

End Sub

```

Public Function GetOutput(n As Integer, Inp As Range)
    Dim i As Integer, j As Integer
    If (Not WeightsLoaded) Then Call Load
    If (Not StateAllocated) Then Call StateAllocate
    For i = 1 To NH
        State1(i) = 0
        For j = 1 To NI
            State1(i) = State1(i) + W1(i, j) * Inp.Cells(1, j).Value
        Next j
        State1(i) = State1(i) + W1(i, NI + 1)
        State1(i) = State1(i) / (1 + Abs(State1(i)))
    Next i
    For i = 1 To NO
        State2(i) = 0
        For j = 1 To NH
            State2(i) = State2(i) + W2(i, j) * State1(j)
        Next j
        State2(i) = State2(i) + W2(i, NH + 1)
        State2(i) = State2(i) / (1 + Abs(State2(i)))
    Next i
    GetOutput = State2(n)
End Function

Function frange(i As Integer, r As Range)
    frange = r.Cells(1, i).Value
End Function

Sub test()
    Dim r As Range
    Set r = Sheets("Simulation").Range("B4:C4")
    Sheets("Simulation").Cells(4, 6) = GetOutput(1, r)
    Set r = Sheets("Simulation").Range("B5:C5")
    Sheets("Simulation").Cells(5, 6) = GetOutput(1, r)
    Set r = Sheets("Simulation").Range("B6:C6")
    Sheets("Simulation").Cells(6, 6) = GetOutput(1, r)

```

```

Set r = Sheets("Simulation").Range("B7:C7")
Sheets("Simulation").Cells(7, 6) = GetOutput(1, r)
End Sub

```

A.10 sdkvars.bat

```

Set Path=C:\Program
Files\Microsoft.NET\SDK\v1.1\Bin\;C:\WINDOWS\Microsoft.NET\Framework\v1.1.4
322\;C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\bin\;C:\Program
Files\Microsoft Visual Studio .NET 2003\Common7\IDE\;%PATH%
Set LIB=C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\lib\;C:\Program
Files\Microsoft.NET\SDK\v1.1\Lib\;%LIB%
Set INCLUDE=C:\Program Files\Microsoft Visual Studio .NET
2003\Vc7\include\;C:\Program
Files\Microsoft.NET\SDK\v1.1\include\;%INCLUDE%
Set NetSamplePath=C:\PROGRA~1\MICROS~1.NET\SDK\v1.1\

```

A.11 battery_dll.bat

```

call sdkvars.bat
csc /out:Battery.dll /target:library Battery.cs
pause

```

A.12 TrainRules.bat

```

call sdkvars.bat
csc /out:TrainRules.exe /target:exe /reference:Network.dll TrainRules.cs
pause

```

A.13 FR_COMPILE.bat

call sdkvars.bat

csc /out:CCont.exe /target:exe /reference:Battery.dll /reference:Network.dll

FR_ChargingControl.cs

pause

A.14 CCont.bat

CCont.exe >res.txt

A.15 dll.bat (for Network.dll)

call sdkvars.bat

csc /out:Network.dll /target:library /reference:FILEIO.dll Network_D.cs Genetic_D.cs

pause

A.16 COMPILE2.bat

call sdkvars.bat

csc /out:CCont.exe /target:exe /reference:Network.dll ChargingControl.cs

pause

Appendix B. Publications by the Candidate Relevant to the Thesis

B.1 Symposium & Conference Proceedings Publications

- [1] Perviz Ali-zade, Kaan Uyar, "*A New Approach for Portable Computer Accumulator Battery Life Saving Charging*", Proceedings of the 2nd International Electrical, Electronics & Computer Engineering Symposium (NEU-CEE'2004), pp. 334-338 Nicosia, TRNC, March 2004.
- [2] Perviz Ali-zade, Kaan Uyar, "*A Fuzzy Controller Approach to Accumulator Battery Life Saving Electrical Desulfatation*", Proceedings of the International Conference on Computational Intelligence (ICCI'2004), pp. 143-147, Nicosia, TRNC, May 2004.
- [3] Perviz Ali-zade, Kaan Uyar, "*Ni-Cd Battery Fuzzy Controlled Charger Rules Based on Internal Resistance Data*", Proceedings of the 3rd FAE International Symposium, pp. 369-373, Lefke, TRNC, November 2004.
- [4] Kaan Uyar, "*Modeling of Nickel Cadmium Batteries*", Proceedings of the 3rd International Conference on Soft Computing, Computing with Words and Perceptions in System Analysis, Decision and Control (ICSCCW-2005), pp. 263-268, Antalya, Turkey, Sep. 2005.
- [5] Kaan Uyar, "*Fuzzy-Neural Control System for Nickel Cadmium Battery Charging*", Proceedings of the 3rd International Conference on Soft Computing, Computing with Words and Perceptions in System Analysis, Decision and Control (ICSCCW-2005), pp.168-178, Antalya, Turkey, Sep. 2005.
- [6] Kaan Uyar, "*Modeling and Simulation of NiCd Batteries Behavior Under Fast Charging with Genetic Algorithm Based Trained Recurrent Fuzzy Neural Network*", Proceedings of the 7th International Conference on Application of Fuzzy Systems and Soft Computing (ICAFS – 2006), Siegen, Germany, Sep. 2006
- [7] Kaan Uyar, "*A Novel Intelligent Approach for Modeling and Control of Non-linear Dynamic Systems*", *Proceedings of the 3rd International Electrical, Electronics & Computer Engineering Symposium 2006*, pp. 291-296, Nicosia, TRNC, Nov. 2006.

B.2 Journal Publications

- [1] Kaan Uyar, "*Recurrent Fuzzy Neural Network Based Modeling and Control of Nickel Cadmium Batteries Charging*", *Journal "Knowledge" ("Technical Sciences")*, vol. 3-4, pp. 6-16, Azerbaijan, 2005.

B.3 Pending

[1] Rafik Aliev, Rashad Aliev, Babek Guirimov, Kaan Uyar, "Dynamic Data Mining Technique for Battery Charging Rules Extraction", *Applied Soft Computing*, Elsevier.

[2] Rafik Aliev, Rashad Aliev, Babek Guirimov, Kaan Uyar, "Recurrent Fuzzy Neural Network Based System for Battery Charging", *ISNN 2007, LNCS*, June 2007.