AN EVOLUTIONARY PLATFORM FOR RETARGETABLE IMAGE AND SIGNAL PROCESSING APPLICATIONS

by

GORN TEPVORACHAI

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Dissertation Advisor: Christos A. Papachristou, PhD

Department of Electrical Engineering and Computer Science
CASE WESTERN RESERVE UNIVERSITY

August 2008
AN EVOLUTIONARY PLATFORM FOR RETARGETABLE IMAGE AND SIGNAL PROCESSING APPLICATIONS

CASE WESTERN RESERVE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

We hereby approve the dissertation of

GORN TEPVORACHAI

candidate for the Doctor of Philosophy degree. *

Committee Chair_________________________________ Chris Papachristou, PhD
Dissertation Advisor, Professor
Department of Electrical Engineering and Computer Science

Committee________________________________________ Daniel Saab, PhD
Department of Electrical Engineering and Computer Science

Committee________________________________________ Frank Merat, PhD
Department of Electrical Engineering and Computer Science

Committee________________________________________ Vira Chankong, PhD
Department of Electrical Engineering and Computer Science

Committee________________________________________ Frank Wolff, PhD
Department of Electrical Engineering and Computer Science

Date: April 11, 2008

*We also certify that written approval has been obtained for any proprietary material contained therein.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Contents</strong></td>
<td>1</td>
</tr>
<tr>
<td>List of Figures</td>
<td>8</td>
</tr>
<tr>
<td>List of Tables</td>
<td>12</td>
</tr>
<tr>
<td>Abstract</td>
<td>14</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>15</td>
</tr>
<tr>
<td>1.1 Overview</td>
<td>15</td>
</tr>
<tr>
<td>1.2 Motivation</td>
<td>16</td>
</tr>
<tr>
<td>1.3 Summary of Project Approach</td>
<td>19</td>
</tr>
<tr>
<td>1.4 Research Questions and Claims</td>
<td>22</td>
</tr>
<tr>
<td>1.5 Research Scope</td>
<td>25</td>
</tr>
<tr>
<td>1.6 Potential Applications</td>
<td>26</td>
</tr>
<tr>
<td>1.7 Thesis Organization</td>
<td>27</td>
</tr>
<tr>
<td><strong>2 Background</strong></td>
<td>30</td>
</tr>
</tbody>
</table>
2.1 Computational Intelligence ........................................... 30
  2.1.1 Artificial Neural Network ....................................... 35
  2.1.2 Neural Net Training .............................................. 37
  2.1.3 Neural Net Ensembles ........................................... 40
2.2 Evolution through Configurability .................................. 41
  2.2.1 Reconfigurable Systems .......................................... 43
  2.2.2 Retargetable Information Processing Applications .......... 45
  2.2.3 Hardware/Software System Configurability .................. 47
2.3 Information Cognitive Modeling .................................... 50
  2.3.1 Cognitive Models ................................................ 50
  2.3.2 Machine Learning ................................................. 52
  2.3.3 Multi-Sensory Information ....................................... 57

3 Cognitive Processing on Evolutionary Platform ....................... 58
  3.1 Cognitive Processing Overview .................................... 58
    3.1.1 Target Profiles and Signatures ................................. 59
    3.1.2 Cognitive Processing Platform Architecture ................ 60
    3.1.3 Technical Approach ............................................ 60
  3.2 Sensory Processing (Phase One) .................................. 62
  3.3 Signature Fusion (Phase Two) ..................................... 65
  3.4 Cognitive Recall (Phase Three) ................................... 70
3.5 Evolutionary Learning (Training Scheme) .................................... 72

3.5.1 Preparation ................................................................. 74

3.5.2 Nested-Loop Training .................................................... 76

3.5.3 Definitions and Notations ................................................. 77

3.6 Coefficient Training (Inner Loop) .......................................... 78

3.7 Structural Evolution (Middle Loop) ........................................ 79

3.7.1 Structural Evolution Growth and Decay ................................. 81

3.7.2 Structural Evolution Parameters ........................................ 82

3.7.3 Structural Evolution Decisions .......................................... 85

3.7.4 Structural Evolution Training ............................................ 87

3.8 Training Data Enrichment (Outer Loop) .................................... 88

3.8.1 Adding/Removing Criterion ............................................. 89

3.8.2 Design Rational ............................................................. 90

4 Self-Configurable Neural Net Processor for FIR Filter Applications ...... 92

4.1 Introduction and Motivation ................................................... 93

4.2 FIR Filter and Neural Net ..................................................... 95

4.2.1 Finite Impulse Response Filters ....................................... 95

4.2.2 Neural Net as FIR Filter ................................................ 96

4.2.3 Neural Net as Configurable, Adaptive Filter ......................... 98

4.3 The Self-Configurable Approach ........................................... 98
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3.1 Coefficient Adjustment Process</td>
<td>100</td>
</tr>
<tr>
<td>4.3.2 Structural Modification Process</td>
<td>102</td>
</tr>
<tr>
<td>4.3.3 Terminal Condition</td>
<td>104</td>
</tr>
<tr>
<td>4.4 Experimentation Results</td>
<td>106</td>
</tr>
<tr>
<td>4.4.1 Noise Cancellation</td>
<td>106</td>
</tr>
<tr>
<td>4.4.2 Signal Equalization</td>
<td>111</td>
</tr>
<tr>
<td>4.4.3 Comments on the Results</td>
<td>113</td>
</tr>
<tr>
<td>4.5 Discussion and Analysis</td>
<td>114</td>
</tr>
<tr>
<td>5 Configurable FIR Filter Scheme based on an Adaptive Multilayer Network Structure</td>
<td>115</td>
</tr>
<tr>
<td>5.1 Introduction and Motivation</td>
<td>116</td>
</tr>
<tr>
<td>5.2 Background and Related Works</td>
<td>118</td>
</tr>
<tr>
<td>5.2.1 Traditional FIR Filters</td>
<td>118</td>
</tr>
<tr>
<td>5.2.2 Transposed and Hybrid FIR Filter Architectures (Related Works)</td>
<td>120</td>
</tr>
<tr>
<td>5.2.3 Multilayer Network Structure as FIR Filters (Contributions)</td>
<td>121</td>
</tr>
<tr>
<td>5.3 Formulation of MLNS as FIR Filters</td>
<td>123</td>
</tr>
<tr>
<td>5.4 The MLNS Architecture</td>
<td>126</td>
</tr>
<tr>
<td>5.4.1 Implementation at MAC Level</td>
<td>126</td>
</tr>
<tr>
<td>5.4.2 Implementation at Neural Cell Level</td>
<td>127</td>
</tr>
<tr>
<td>5.4.3 Implementation at Neural Layer Level</td>
<td>128</td>
</tr>
</tbody>
</table>
5.5 Complexity Analysis ................................................. 128
   5.5.1 MAC Usage, Latency, and Design Throughput Analysis . . 129
   5.5.2 Sample Synthesis Result ..................................... 132
   5.5.3 Sample Simulation Result ................................... 134

5.6 Potential Applications ............................................ 137

5.7 Discussion and Analysis .......................................... 139

6 Multi-Label Imbalanced Data Enrichment Process in Neural Net Classifier Training ................................................. 141

   6.1 Introduction ...................................................... 142

   6.2 Background ..................................................... 144
       6.2.1 Related works ........................................... 144
       6.2.2 Overview of the Enrichment Process .................. 145

   6.3 Our approach .................................................... 147
       6.3.1 Enrichment Initialization ............................... 147
       6.3.2 Enrichment Update ..................................... 149
       6.3.3 Enrichment Termination ............................... 152

   6.4 Experimentation ................................................ 153
       6.4.1 Setup ................................................... 155
       6.4.2 Results and Measurements ............................. 156
       6.4.3 Analysis and Discussion ............................... 161
7 Facial Image Associative Memory Model ........................................163

7.1 Introduction and Motivation .................................................... 164

7.2 Background ............................................................................. 165

7.2.1 Classical Associative Memory ............................................. 165

7.2.2 Related Works ..................................................................... 167

7.2.3 Our Contributions ................................................................. 168

7.3 Facial Image Associative Memory Design ................................. 169

7.3.1 Sensory Pre-processing ....................................................... 170

7.3.2 Interfusion and Association Phases .................................... 173

7.3.3 Augmented Face Detection ................................................. 174

7.4 Experimental Simulation Results ........................................... 176

7.5 Discussion and Analysis ......................................................... 179

8 Conclusion and Future Works .................................................... 181

8.1 Discussions and Remarks ....................................................... 181

8.1.1 Remarks on Neural Net Ensemble ................................. 181

8.1.2 Evolution and Machine Learning .................................. 183

8.1.3 Information Integration .................................................. 184

8.2 Summary of Research Contributions .................................. 185

8.3 Limitations .............................................................................. 189

8.4 Publications ............................................................................ 189
List of Figures

2-1  Reconfigurable system model  44
2-2  Application design to computational instruction and data struc-
ture translation  46
2-3  Cellular and polymorphic architectures cooperative integration  47
3-4  Evolutionary system architecture  61
3-5  Cognitive processing (Classification/Identification)  62
3-6  Sensory processing with pre-processors and feature extractors  63
3-7  Signature fusion with feature interfusors  66
3-8  Cognitive recall with signature associators and knowledge base  72
3-9  Autonomous learning scheme layout with central evolutionary ser-
vice  73
3-10 Evolutionary learning (training scheme)  76
3-11 Coefficient training process  79
3-12 Overall evolutionary service protocol of system development  83
3-13 Structural evolution  84
<table>
<thead>
<tr>
<th>Page</th>
<th>Section Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-14</td>
<td>Four separated neural nets for evolution decisions</td>
<td>85</td>
</tr>
<tr>
<td>3-15</td>
<td>Dedicated neural net for input tap parameter evolution decision</td>
<td>85</td>
</tr>
<tr>
<td>3-16</td>
<td>Training data enrichment process</td>
<td>90</td>
</tr>
<tr>
<td>4-17</td>
<td>Direct-form FIR filter structure model equivalent to transfer function in Equation 4-5</td>
<td>96</td>
</tr>
<tr>
<td>4-18</td>
<td>Direct-form FIR filter structure model implemented by linear two-layer perceptron</td>
<td>97</td>
</tr>
<tr>
<td>4-19</td>
<td>Neural net processor training operation flow</td>
<td>99</td>
</tr>
<tr>
<td>4-20</td>
<td>Noise cancellation model</td>
<td>107</td>
</tr>
<tr>
<td>4-21</td>
<td>Neural net processor training model for noise cancellation application</td>
<td>108</td>
</tr>
<tr>
<td>4-22</td>
<td>Noises and contaminated signals</td>
<td>109</td>
</tr>
<tr>
<td>4-23</td>
<td>Neural net processor training model for signal equalization application</td>
<td>111</td>
</tr>
<tr>
<td>5-24</td>
<td>Direct-form FIR filter structure equivalent to transfer function in Equation 5-15</td>
<td>119</td>
</tr>
<tr>
<td>5-25</td>
<td>Full scale multilayer network structure (MLNS) model</td>
<td>122</td>
</tr>
<tr>
<td>5-26</td>
<td>Network cell (NC) model</td>
<td>123</td>
</tr>
<tr>
<td>5-27</td>
<td>Serial neural cell (NC) architecture</td>
<td>127</td>
</tr>
<tr>
<td>5-28</td>
<td>Folded network layer (NL) architecture</td>
<td>129</td>
</tr>
</tbody>
</table>
5-29 Hardware complexity comparison for the six filter implementations 133
5-30 Throughput comparison for the six filter implementations . . . . . 133
5-31 MLNS training model for adaptive filter application . . . . . . . . 136
6-32 Pseudo code for enrichment process . . . . . . . . . . . . . . . . 148
6-33 Sampled imbalanced control data (multi-label) [note: blue “○” represents robot arm in state 0; red “□” represents state 1; and magenta “♦” represents state 2] . . . . . . . . . . . . . . . . . . . . . 154
6-34 Same sampled imbalanced control data as in Figure 6-33 in 3-dimensional plot (multi-label) . . . . . . . . . . . . . . . . . . . . . . . . . . . 154
6-35 Initial enrichment active training set at enrichment iteration 1 [note: the dotted splines indicate 3 initial clusters.] . . . . . . . . . . . . . . 156
6-36 Enrichment training data update at iteration 4 . . . . . . . . . . . 158
6-37 Number of training data points in high-error, middle-error, and low-error groups . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 159
6-38 Number of training data points being added and removed . . . . . 159
7-39 Abstract associative memory model . . . . . . . . . . . . . . . . . 166
7-40 Facial image associative memory model three-phase process . . . 169
7-41 Multi-sensor simultaneous pre-processing operations . . . . . . . 171
7-42 New sensory pre-processing with preferential/imprecise transforma-
    tion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 173
7-43 Image associative memory model in large scale . . . . . . . . . . 175
7-44 Training energy error progress . . . . . . . . . . . . . . . . . . . . . 178

7-45 Testing energy error for each testing image (as testing index) . . . . 179

7-46 Image correlation for each testing input-target image pair (as testing index) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 179
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>Standardized terminologies in data fusion technologies from [1]</td>
<td>68</td>
</tr>
<tr>
<td>3-2</td>
<td>Information needed for parameter evolution decisions</td>
<td>86</td>
</tr>
<tr>
<td>4-3</td>
<td>Result of noise cancellation neural net processor learning</td>
<td>110</td>
</tr>
<tr>
<td>4-4</td>
<td>Noise cancellation on ideal traditional adaptive FIR filter configuration</td>
<td>110</td>
</tr>
<tr>
<td>4-5</td>
<td>Result of signal equalization neural net processor learning</td>
<td>112</td>
</tr>
<tr>
<td>4-6</td>
<td>Signal equalization on ideal traditional adaptive FIR filter configuration</td>
<td>113</td>
</tr>
<tr>
<td>5-7</td>
<td>MAC usage, latency, and design throughput analysis comparison</td>
<td>131</td>
</tr>
<tr>
<td>5-8</td>
<td>Result of low pass (LP), high pass (LP), band pass (BP), and band stop (BS) adaptive filters</td>
<td>136</td>
</tr>
<tr>
<td>6-9</td>
<td>Neural net performance over enrichment training process</td>
<td>158</td>
</tr>
<tr>
<td>6-10</td>
<td>Active training set distribution and training data mean (average)</td>
<td>161</td>
</tr>
</tbody>
</table>
6-11 Comparison neural net performance at iteration 80 [* is the average distribution over the enrichment process in Table 6-10] . . . . . 161

7-12 Face mask relative positions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 176
An Evolutionary Platform for Retargetable Image and Signal Processing Applications

Abstract

by

GORN TEPVORACHAI

In this thesis, we propose a cognitive information processing system (cognitive processing) on an evolutionary platform for retargetable applications such as facial image recognition, image feature extraction, evolvable filters, and environmental information tracking. Cognitive processing can process multi-sensory information on an automated system such as an unmanned vehicle or a surveillance unit in a remote site to avoid harsh terrain. Evolutionary platform supports the ability to change information processing behavior to comply with ever changing environment in order to accomplish a mission objective. The cognitive processing model can overcome particular difficulties to traditional search, exploration, and engineering decision making applications. The proposed cognitive strategies emphasize the decomposition of multi-sensory information, the re-construction of internal representations, and the cognitive processing of combined information which yield sub-optimal solutions and indicate best local system direction. Several applications, such as facial image recognition and digital signal processing, are used to verify our models and compare them with other well-known approaches. The derived simulation and synthesized results show that the proposed cognitive processing model on evolutionary platform attains better performance than those of the conventional methods.
Chapter 1

Introduction

1.1 Overview

Current information computing systems can be characterized as static in nature. They rely on fixed architectures and specific software optimizations [2]. Today's computing systems were developed for fixed mission scenarios and cannot provide effective information processing capability to support retargetable mission applications. They are not able to accommodate growing past experience on dynamic collaborative information-centric strategies. The lack of versatility to dynamic mission requirements results in reduced performance or poorly matched processing of the application. A unique processing design for each specific mission sensor configuration cannot be afforded due to the required cost for multiple platforms and the inability to accurately define and predict mission variations before the deployment.

The majority of the existing works are focused on specific evolutionary
system components, indirectly investigating possible system integration. For example, the IBM research center has developed a cellular computing architecture [3] for dual/quad core computer processor. This work is concentrated on reconfigurable architectures, which can potentially be the platform of choice for information processing and evolutionary system. Various algorithms and technologies rely on stochastic modeling of evolutionary techniques [4, 5, 6, 7] for some possibility that an innovative design will emerge. Their approaches do not guarantee the emergence of a novel design or easily reproduction of the design at a later time.

In this thesis, we propose a cognitive information processing (cognitive processing) on an evolutionary platform for retargetable applications. The main contributors of this work are the cognitive processing (neural net ensembles) and the evolutionary platform (evolutionary training scheme). Our model considers multiple sensory information and generates the best cognitive recall from the memory. The proposed cognitive processing has been demonstrated in facial image recognition, image feature extraction, evolvable filter and other applications [8, 9, 10, 11, 12].

1.2 Motivation

An evolutionary system can be viewed as a system which has an ability to change its behavior to comply with its environment (external stimuli). It can re-organize its functionalities and structures with respect to external stimuli without human
operators. A wide spectrum of engineering applications is presented and particular attention is paid to automated mission accomplishment problem areas that can be found in many engineering applications. Some applications for automated mission accomplishment are intended for deployment in remote sites, harsh terrain for human exploration, or lack of human assistance, such as an unmanned vehicle aircraft, a deep space explorer, or a surveillance camera/video unit. The evolutionary system has potential benefits on a remote site robot or in harsh environment, such as a deep space satellite, an unmanned aviation device, or a deep sea explorer. Instead of risking human lives on the high risk environment or requiring human resource, evolutionary systems can perform similar routine tasks. Other problems in these areas include image feature extraction, evolvable filter, preference learner, collaborative data, space explorer, and associative memory.

However, the evolutionary system needs to be well-trained and specific mission directed. This support is in the form of the provision of relevant information not only pertaining to an identified optimal solution for a specific condition but also to various characteristics of the system behavior. For instance, extracted information may relate to parameter sensitivity, degree of constraint satisfaction and multi-objective considerations. Whenever the target application mission objective changes, evolutionary system should be able to develop a mechanism to switch its surviving strategies. The mechanism can both increase performance while highlighting shortfalls of the mathematical models describing the system behavior under design.
Sometimes remote access communications between the device and the base control are limited. This proposed cognitive information processing system (cognitive processing) on evolutionary platform can overcome particular difficulties that present severe problems to more stand-alone task specific systems while also providing significant support to the engineering decision making process. The routine tasks required for such vehicles may involve regular parameter monitoring and decision making when the parameters reach beyond pre-specified limits. Therefore, the cognitive processing can potentially substitute the human controls either on-site or remotely. The cognitive processing will have an ability of recognizing its environment conditions and adapting itself to the current conditions; potentially, it will be able to discover new sets of operations to improve its survivability. An emphasis, therefore, is on the extraction of knowledge from the mission objective description while also identifying optimal solutions which indicate best system direction.

The development of this proposed cognitive processing involving various techniques, including adaptive processing and evolutionary approaches, which can provide extensive support in multiple mission application design. Applied and fundamental researches are required to ensure the best integration of available computing solutions by improving the overall efficiency of various existing techniques and promoting the new evolutionary system designs. Complex issues relating to the integration of application requirements with adaptive processes and the interactive role of adaptive processes within evolutionary system must also be considered. This area of widening adaptive computing potential within
engineering is now receiving significant attention from several international research groups and prototype cooperative artificial intelligence and evolutionary tools are gradually emerging [13, 14, 15].

Moreover, the cognitive processing should be more accessible to engineers and researchers via prototype application examples and strategies that address retargetable information processing problem areas observed in many applications. This will promote awareness for powerful information processing capabilities and retargetable multi-mission design for various applications such as image feature extraction, evolvable filter, collaborative data, preference learner, space explorer, and associative memory. The proposed conceptual framework, in this work, incorporates a broader view of information processing and evolutionary system mechanisms than single artificial intelligence, machine learning, optimization technique, or an application specific architecture.

### 1.3 Summary of Project Approach

A computation intelligence approach, such as artificial neural nets and evolutionary computation techniques, can overcome the previous limitations. It has been shown mathematically that an artificial neural net is capable of learning continuous non-linear input-output relationships [16]. A common criticism for neural net-based applications is the tedious process of selecting the proper neural net architecture [17]. In general, it is not easy to choose the right neural net
topology for a particular application. The design of such neural net involves determination of the following:

1. The number of neural net layers.

2. The number of neurons in each layer.

3. The weights of the connections between neurons.

In traditional neural nets, their design depends mainly upon the human operators experience and the trial and error training process. Moreover, it has become apparent that there are many classification tasks which cannot be effectively solved by means of training simple unitary neural net [18].

The main innovation of our method is the cognitive information processing scheme consisting of three phases:

1. Multi-sensory information filtering.

2. Fusion of the sensory information associated with a target.

3. Cognitive type of matching of the input sensory information to similar targets pre-stored in the memory or in the knowledge base.

The key element of our method is the use of novel evolutionary technique for offline learning on the proposed evolutionary platform and training of realistic targets under several experimental scenarios. The learning includes the following three iterative loops: coefficient training, structural evolution, and enrichment process.
A very large collection of academic literatures describing computational intelligence \[19, 6, 20, 21, 22\], evolutionary techniques \[23, 5\], and information cognitive models \[24, 25, 26\] that address particular aspects relating to evolutionary system performance across a wide range of multivariable problems is now available. This collection of solutions and technologies may discourage engineers and researchers who begin to explore the possibilities for the new techniques.

The proposed cognitive processing introduces a paradigm shift from static fixed architecture and specific software optimization to address retargetable information processing schemes, application software, and hardware implementation. The conceptual framework establishes common understanding for the purpose of this work by providing the mathematical analysis and prototypes of sensory processing, signature fusion, cognitive recall, coefficient training, structural evolution, and enrichment process. Our main arguments concern two operations: the cognitive processing (including signal processing, signature fusion, and cognitive recall) and the evolutionary platform (including coefficient training, structural evolution, and enrichment process). Together, these operations illustrate important characteristics in different types of retargetable information processing applications. The fundamental concepts of these two operations are explained with the usage of computational intelligence, evolutionary technique, and information cognitive model. Mathematical equations provide more accuracy to the cognitive processing on evolutionary platform description especially when they are complex computational algorithms.
A set of measurement metrics have been developed to support system design and optimization to include algorithmic computation complexity (time) and storage memory used during normal operation and training mode. Once the cognitive processing implementation is complete, the following verification and validation will serve as system test bench and proof-of-concept for experimental and commercial applications with respect to the measurement metric.

1.4 Research Questions and Claims

This section points out our open research questions regarding the cognitive processing, its components, evolutionary platform, and retargetable information processing applications. Each research question will be addressed by a series of designs, illustrations, and mathematical equations in further investigation.

**Questions:**

- What are the benefits and limitations of cognitive processing based on evolutionary platform.

- What are the benefits and limitations of evolutionary platform within the design space of retargetable information processing applications.

- What are the roles and purposes of sensory processing, signature fusion, and cognitive recall in information processing application.
• What are the benefits and limitations of sensory processing, signature fusion, and cognitive recall in information processing application.

• What are the roles and purposes of coefficient training, structural evolution, and enrichment process in evolutionary platform.

• What are the benefits and limitations of coefficient training, structural evolution, and enrichment process in evolutionary platform.

• How the cognitive processing and the evolutionary platform interact to improve the performance of retargetable information processing applications.

• What are potential retargetable information processing applications to deploy the cognitive processing.

• What are the benefit and limitations of the cognitive processing model in the potential retargetable applications.

• What measurements can be used to verify, validate, and benchmark the cognitive processing in the potential applications.

From our research investigations, experiments, and results, the cognitive processing on evolutionary platform exhibits the following utilities for retargetable information processing applications. Each claim will be verified by a series of designs, illustrations, and mathematical analysis in this work.

Claims:
• The proposed cognitive processing has the capability to provide optimized sensory processing implementations without the necessity of customized development effort for each specific application objective, including associated time and cost.

• The proposed cognitive processing is applicable to wide range of application objectives and sensor suites. It is optimized for each specific application objective and/or sensor suite providing for tactical and strategic opportunities as well as technical upgradability.

• The proposed cognitive processing has the ability to adapt and optimize its parameters and structures based on provided information using a collection of artificial intelligence and machine learning techniques in coefficient training, structural evolution, and enrichment process regardless of the data characteristic.

• The evolutionary platform for neural net training (includes coefficient training, structural evolution, and enrichment process) provides the ability to re-target application objectives and sensory suites regardless of application dynamics and sensory information characteristic.

• The proposed cognitive processing can reduce the adaptation, optimization, and verification time and effort from years, in many potential applications, to days or minutes.
1.5 Research Scope

It is important to be clear about the scope and boundaries of this work. This work is theoretic (conceptual) and implemental (practical) focusing on the proposed cognitive information processing on evolutionary platform for retargetable applications. As indicated in the objectives and goals, research questions, research claims, and project approach, our studies address issues of principle empirical observations about computational intelligence [19, 6, 20, 21, 22], evolutionary techniques [23, 5], information cognitive model [24, 25, 26], and verification and test bench for retargetable information processing applications.

This work does not claim that the cognitive processing mechanism provides a general purpose problem solving algorithm. However, the work is useful in possible computational methods and in expansion and understanding of existing computational methods. It provides specific algorithmic components and specialized algorithms for the purposes of illustrating retargetable information processing application.

The work does not require that the evolutionary platform we define be equivalent to any particular adaptive system in engineering domains and natural domains. However, our models of problem classes are useful for expanding our notions of evolutionary difficulty, and system difficulty in general. We discuss the main limitations and the requirements of different general retargetable information processing application classes.

The conclusions derived from our cognitive processing on evolutionary
platform models should be transferred to natural evolution only with extremely
careful qualification. We do not claim the availability of the necessary conditions
for compositional evolution in nature—this is an empirical matter. Abstract com-
putational evolutionary system models, such as those we use, are relevant to nat-
ural evolution only in the sense that we demonstrate the possibility for biological
evolution processes to behave in the manner reported in this work. However,
our models assist us in identifying what those conditions might be and provide
a description of the features that may be examined in natural systems. In the
meantime, the conceptual and theoretic principles of the work stand indepen-
dently.

1.6 Potential Applications

Common application areas that can benefit from the proposed cognitive pro-
cessing have been identified as facial image recognition, image feature extrac-
tion, evolvable filter, collaborative data, preference learner, space explorer, and
associative memory. Although potential solutions for each of these intended
applications have been realized to certain efforts, the solutions are specific for
the target applications which, most of the time, are not interchangeable among
these applications. The potential of the proposed evolutionary platform in this
research area goes beyond the optimization for specific problems.
The following chapters discuss the potential applications of cognitive processing to a number of well-defined engineering problems. The generic application areas might include system identification, information decomposition and extraction, constraint satisfaction and optimization, multi objective satisfaction and optimization, adaptive agent policy search strategy, and auto associative memory design. [27, 28, 8, 9]

1.7 Thesis Organization

In the subsequent chapters of this thesis, we will discuss our background, related works, assumptions, and methodology for cognitive processing on evolutionary platform and potential retargetable information processing applications. Chapter 2 provides background description to computational intelligence, evolutionary techniques, and information cognitive models. The chapter concentrates on the applications of neural net in cognitive processing. This includes a discussion of reconfigurable systems, the configurability concept, and evolutionary systems. This chapter also describes the preliminary reviews on information cognitive model and fusion as an approach to the cognitive processing.

Chapter 3 describes the proposed cognitive processing scheme including the description of process sub-modules (sensory processing, signature fusion and cognitive recall) as evolutionary neural net ensembles. The chapter then discusses the evolutionary platform sub-models, including the coefficient training, structural evolution, and enrichment process. This chapter describes the
approach to cognitive processing using neural net ensembles and their training method. The chapter also describes the structural evolution on the neural net in cognitive processing. The chapter then concludes with a discussion of the way which information might be represented by neural net through training data enrichment.

Chapter 4 illustrates an application of neural net coefficient training and structural evolution for adaptive finite impulse response filter. This includes a discussion of the filter design, limitation, and some insights given by experiments in noise cancellation and signal equalization.

Chapter 5 presents the implementation of a reconfigurable FIR filter based on an adaptive multilayer network structure on an FPGA. The chapter also compares the performance of the straight pipelined and the traditional folded filter implementations with several adaptive FIR filter configuration implementations.

Chapter 6 provides the proposed training data enrichment process in detail with an application of an adaptive robot arm controller. The chapter also includes the results of the application accuracy in recognizing robot arm controller states.

Chapter 7 illustrates a facial associative memory model as another information processing application to the cognitive information processing in its full potential. The chapter presents a memory model, experimentation, simulation results with realistic data, and its performance analysis.
Chapter 8 presents the conclusion of this work, in terms of the discussion of the cognitive processing on evolutionary platform, as well as the research contributions of the methodology. It also describes the limitations and possible future extension to this work and gives several suggestions of areas of investigation that may result in improvements over the given assumptions and methodology.
Chapter 2

Background

This chapter reviews the relevant literature in computational intelligence, evolution through configurability, and information cognitive modeling. Existing works using traditional techniques are presented. Section 2.1 begins with a brief description of computational intelligence and artificial neural nets. Section 2.2 describes how evolutionary techniques can complement computational intelligence, especially artificial neural nets. Section 2.3 describes existing information cognitive models and how to process information with artificial neural net and evolution through configurability for cognitive behavior.

2.1 Computational Intelligence

Computational intelligence is a successor to artificial intelligence [6, 21, 29, 25]. In order for an evolutionary system to accomplish a mission objective in an environment, artificial intelligence is a way of reaching the goal. Computational
intelligence [19, 6, 20, 21, 22] is a research area focus in intelligent behavior, learning, and adaptation in machines. Research in computational intelligence is concerned with producing machines to automate tasks requiring intelligent behavior. Examples include control, planning and scheduling, the ability to answer diagnostic and consumer questions, handwriting, speech, and facial recognition. It has become an engineering discipline, focused on providing solutions to real life problem, software applications, traditional strategy games like computer chess and other video games.

There are five focus areas in computational intelligence relying on heuristic algorithms: evolutionary computation, neural networks, fuzzy systems, swarm intelligence, and expert systems (good old-fashioned artificial intelligence–GOFAI). They cover a set of search, exploration, and optimization algorithms that rely upon search and heuristic from a population of trial solutions and iterative search around a single point within a complex multivariate problem space. They generally rely upon a number of deterministic and stochastic operators that maintain a high degree of exploration resulting in a broad sampling of available solutions.

1. **Evolutionary Computation** [4, 5, 6, 7] is a subfield of computational intelligence (artificial intelligence) that involves combinatorial optimization problems. It is considered as an artificial implementation of the natural evolution process. In the natural selection, the parents are stochastically paired and a recombination operator is introduced that randomly selects materials from each parent in order to create offspring (genetic operations).
These offspring then represent the members of the new generation. A random mutation operator is also introduced to perform small numbers of random mutations to variable values within the overall population. Mutation injects new information into the process by mutating randomly selected parameter values depending upon some present random mutation probability. This supports diversity and exploration while preventing premature convergence of the system. Evolutionary computation is an optimization technique based on ideas of adaptation and evolution. Evolutionary computation uses iterative progress mimicking the nature, such as reproduction, recombination, mutation, survival of the fittest, and natural selection in population. The population is then selected in a guided random search using parallel processing to achieve the desired end. Evolutionary computation includes evolution strategies and algorithms such as genetic algorithms, evolutionary programming, evolutionary strategy, genetic programming, and learning classifier system [5].

2. **Neural Network** has been used to refer to a network or circuitry of biological networks. The modern usage of the term often refers to artificial neural networks, which are composed of artificial neurons or nodes. Thus, the term neural network has two distinct connotations:

- Biological neural networks [30, 31] are made up of real biological neurons that are connected or functionally-related in the peripheral nervous system or the central nervous system. In the field of neuroscience,
they are often identified as groups of neurons that perform a specific physiological function in laboratory analysis.

- Artificial neural networks [16, 32, 33] are made up of interconnecting artificial neurons (programming constructs that mimic the properties of biological neurons). Artificial neural networks may either be used to gain an understanding of biological neural networks, or for solving artificial intelligence problems without necessarily creating a model of real biological system.

3. Fuzzy Systems [34, 35, 36, 22] are systems based on fuzzy logic. A fuzzy system is a mathematical system that analyzes analog input values in terms of logical variables that take on continuous values between 0 and 1. It is in contrast to classical or digital system, which operates on discrete values of either 0 or 1 (true or false). Fuzzy systems possess many of the performance characteristics for search, exploration, and optimization. Existing fuzzy techniques have been established and widely utilized within machine control. Fuzzy system strategy, fuzzy logic [35, 37], is analogous to the fact that the logic involved can deal with fuzzy concepts. For example, some natural concepts cannot be expressed as “true” or “false” but rather “partially true” or “partially false”. The input variables in fuzzy systems are generally mapped into by sets of membership functions similar to fuzzy sets. The process of converting a crisp input value to a fuzzy value is called fuzzification.
4. **Swarm Intelligence** [38, 39, 40] is a computational intelligence (artificial intelligence) technique based around the study of collective behavior in decentralized, self-organized systems, in the context of cellular robotic systems. Swarm intelligence systems are typically made up of a population of simple agents interacting locally with one another and with their environment. Although there is normally no centralized control structure dictating how individual agents should behave, local interactions between such agents often lead to the emergence of global behavior. Examples of systems like this can be found in nature, including ant colonies, bird flocking, animal herding, bacterial growth, fish schooling, locust devastating swarm, cricket crawling band, mediocre human swarmers, and cockroaches nest. Swarm intelligence includes algorithms such as ant colony optimization, particle swarm optimization, and stochastic diffusion search.

5. **Expert Systems** [19, 41, 21] also known as a knowledge-based systems, one of machine learning techniques, are computer programs that contain some of the subject-specific knowledge of one or more human experts. This class of programs was first developed by researchers in artificial intelligence and applied in commercial applications. The most common form of expert systems is made up of a set of rules that analyze information about a specific class of problems, as well as providing mathematical analysis of the problem. Depending on their design, they can recommend a course of user action in order to implement corrections. It is a system that utilizes reasoning
capabilities to reach conclusions. Expert systems have shown a major potential when utilized as artificial intelligence either in a stand-alone manner or cooperating within other intelligence systems [29] through logic programming [42, 43].

2.1.1 Artificial Neural Network

In this work, we are focusing solely on artificial neural networks. Artificial intelligence tries to simulate some properties of neural networks. It aims to build mathematical models of biological neural systems. Artificial neural networks have been applied successfully to speech recognition, image analysis, and adaptive control, in order to construct software agents or autonomous robots. Most of the currently employed artificial neural networks for artificial intelligence are based on statistical estimation, optimization, and control theory.

An artificial neural network (also called a simulated neural network or commonly known as neural network or neural net) is an interconnected group of artificial neurons that use a mathematical or computational model for information processing based on a connectionist approach to computation. In most cases, a neural net is an adaptive system that changes its structure based on external or internal information that flow through the network.

Neural nets [44, 16, 32, 18] are non-linear or linear statistical data modelings or decision making tools. They can be used to model complex relationships and functions between inputs and outputs or to find patterns in data. A neural
net involves a network of simple processing elements (artificial neurons) determined by the connections between the processing element parameters. Examples of classical neural net type include Hopfield nets, Perceptrons, and multi-layer Perceptrons. A neural net does not have to be adaptive; however, its practical use comes with algorithms designed to alter the strength (weights) of the connections in the network to produce a desired output.

The utility of neural net models lies in the fact that they can be used to infer a function from observations. This is particularly useful in applications where the complexity of the data or task makes the design of such function by hand impractical. Many statistical regression models that approximate the relationship between the inputs and outputs can be computationally expensive. For instance, a neural net can contribute to the generation of improved mathematical representations of engineering systems when the exact model is unknown. In other words, the neural net provides an adaptation process for a collection of functions to approximate a particular function.

Applications for neural net are wide spread. Function approximation, regression analysis, time series prediction, and black-box modeling [45, 46] concern an estimated or interpolated representation of an unknown mathematical model. Classification, pattern and sequence recognition, novelty detection, and sequential decision making concern with placing individual items into groups based on quantitative information on one or more characteristic inherent in the
items. Data processing, filtering, clustering, blind signal separation, and compression concern the conversion of raw data into information or knowledge. Computationally, inexpensive representations are required which will result in the identification of optimal solutions within an acceptable period of time.

Other application areas include system identification and control (vehicle control, process control), game-playing and decision making (backgammon, chess, racing), pattern recognition (radar systems, face identification, object recognition and more), sequence recognition (gesture, speech, handwritten text recognition), medical diagnosis, financial applications, data mining (or knowledge discovery in databases, “KDD”), visualization and e-mail spam filtering [47].

2.1.2 Neural Net Training

Neural net training requires a set of input-target pair data, prior knowledge of input and target information, and the initial guess structure of the unknown model. This adaptive learning method takes advantage of assessing the errors between input-target pairs and re-adjusting the neural net application. After a certain amount of iterations of the adaptive learning algorithm, if the approximation performance of the neural net improves too slowly or has been over trained, the training will attempt to adjust the coefficients and/or the structure of the neural net. The resulting neural net representation will contribute to a significant
reduction in computational cost when utilized as an approximation to the eval-
uation function. Some examples of neural net is image feature extraction, evolv-
able filter, collaborative data, preference learner, space explorer, and associative
memory.[48]

Providing a set of input data and a set of desired data, the neural net can it-
erate through evaluation and yield the approximation of the desired data at some
levels of error. Preliminary mathematical models can be coarse representations
of the system under design characterized by mathematical approximation either
due to lack of knowledge, lack of confidence in available data or the requirement
that the computational cost must be kept to a minimum.

**Back-Propagation**

Back-propagation is described in [48, 49]. To perform back propagation on a neu-
ral net, a given training set \((T)\) consisting of sample inputs \((I_k)\) and their corre-
sponding desired outputs \((O_k)\) must be presented. During a training iteration,
many training pairs (sample inputs and desired outputs) are presented to the
neural net. The neural net evaluates the inputs \((x)\) and produces estimated out-
puts \((y)\). The most widely used back propagation scheme uses the Euclidean dis-
tance similarity or the least mean-square algorithm to calculate the gradient of
the net outputs as in Equation 2-1 [50, 51]. The gradient measures the difference
(or similarity) between the estimated output and the desired output of the training data.

\[
\frac{\partial E}{\partial w_{ij}} = (O_{k,j} - y_j) \cdot f'(\text{sum}_j) \cdot x_j \quad 2-1
\]

where \( y_j \) is the \( j \)-th neuron's estimated output, \( O_{k,j} \) is the \( j \)-th desired output and \( f'(\text{sum}_j) \) is the first derivative of the activation function evaluated at the \( j \)-th output neuron's \text{sum} value. The coefficient training updates all net weights as in Equation 2-2

\[
w_{i+1} = w_i - \eta_i \frac{\partial E}{\partial w} = w_i + \eta_i (O_k - y) \frac{\partial y}{\partial w} \quad 2-2
\]

where \( i \) is the iteration index and \( \eta \) is the adaptive learning rate.

**Variable Learning-Rate**

The variable learning-rate, \( \eta \), is adjusted after every training iteration. The adjustment is an attempt to accelerate the speed of back propagation convergence. In this case, the learning-rate update relation is a function of the gradient. The learning rates \( (\eta) \) for all net weights are updated as shown in Equation 2-3

\[
\eta_{i+1} = \eta_i - \frac{\partial E}{\partial \eta} = \eta_i - \frac{\partial E}{\partial w} \cdot \frac{\partial w}{\partial \eta} \quad 2-3
\]

where \( i \) is the iteration index. The \( \frac{\partial w}{\partial \eta} \) is the gradient of the net weight with respect to the learning-rate, which is computed by Equation 2-4

\[
\frac{\partial w}{\partial \eta} = \sum_i \sum_j (O_{k,j} - y_j) \cdot f'(\text{sum}_j) \cdot x_i \cdot \frac{\partial E^-}{\partial w_{ij}} \quad 2-4
\]

where \( \frac{\partial E^-}{\partial w_{ij}} \) is the gradient of the error with respect to the net weight from the previous training iterations.
The coefficient adjustment process initializes the learning-rate to zero at the beginning of the evolutionary training.

### 2.1.3 Neural Net Ensembles

An ensemble, or a committee machine, is a group of learners which are combined together in order to obtain a better generalization than that obtained by a single learner. The errors of the individual ensemble members cancel out to some degree when their predictions are combined. Non-linear models like neural nets and decision trees are examples of such learning algorithms. In this section, the emphasis is on ensembles of neural nets.

Standard practice [52, 17, 16, 53] dictates that we perform some trial tunings to find an acceptable architecture and tuning for the network and then trust all future classifications to the best network we find. It turns out, however, that it is preferable to keep the complete set of the networks (or at least a screened subset) and run them all with an appropriate collective decision strategy. In analogy with physical theory, we will refer to the set of neural nets used as an ensemble.

A neural net ensemble is a very successful technique where the outputs of a set of separately trained neural nets are combined to form one unified prediction. Recently, researchers have investigated the technique of combining the outputs (predictions) of multiple neural net classifiers to produce a single classifier [54, 16, 53, 55]. Many researchers have shown that simply combining the output of many predictors can generate more accurate predictions than that of
any of the individual predictors [56, 57, 58]. The resulting classifier is referred to as an ensemble or a committee machine. In particular, combining separately trained neural nets (commonly referred to as a neural net ensemble) has been demonstrated to be particularly successful [59, 60, 61]. The individual neural nets making up the ensemble are generated via a learning process. The output of the ensemble of a new input is calculated as a combination of the outputs of its individual neural net members. Classification performance of the ensemble is proven to be better than any of its members mathematically [55] as well as experimentally [54].

2.2 Evolution through Configurability

An evolutionary system possesses two key properties: artificial intelligence and autonomous learning. An evolutionary system must be able to change its behavior autonomously to operate within its environment, to re-target its mission in pursuit to accomplish its application objective, or to be capable of making independent decision. Artificial intelligence [6, 21, 29, 25] also plays a major role in an evolutionary system with the support of cellular polymorphic architecture to perform adaptation.

An evolutionary system that can learn and adapt is an excellent evidence for machine intelligence. An evolutionary system can be taught basic concepts, primitive motor skills, how to gather information, navigate, and even how to
teach itself. Learning includes a rich variety of techniques that use previous experience or heuristic information to enable more effective performance. When it comes to designing adaptive behavior, the hardest problem is often the choice of the underlying intelligent mechanism. Some industrial robots must perform the same action, the same way, every day. While such a robot may benefit from an ability to learn this action as efficiently as possible, it may never need to learn a new motion. Other robots, such as a mobile office assistant, may need to learn new information and skills every day. For each task, a designer must decide precisely what should be learned, when learning should occur, the computational means to implement this intelligence, and how much a priori knowledge should be supplied.

In the literature, evolvable hardware is sometimes referred to as self-configurable system or evolutionary system. The emphasis of the evolutionary system is on cellular polymorphic architecture and autonomous learning scheme. The evolutionary system establishes the ability to span a broad dynamic application space by implementing the correlated artificial intelligent modules and the configurable cellular architecture elements. In this work, evolutionary systems refer to reconfigurable systems of hardware, firmware and software that are capable of changing their behaviors by growth or decay in closed-loop operations over a period of time, either short term or over generations. The evolution includes the emergence of new behavior to serve an objective, for example, intrinsic discrete time filters with signal feedback [47].
2.2.1 Reconfigurable Systems

Reconfigurable systems offer both the flexibility of computer software and the ability to construct custom high performance computing hardware. The reconfigurable architecture enables multiple operation modules to be developed in a cooperative environment instead of in stand-alone computer software or offline traditional computing hardware. A reconfigurable system can enhance innovative hardware design and provide architecture support to establish optimal system configuration. It is also possible that cooperative system configuration design between human and evolutionary computing procedures can lead to the emergence of creative solutions. Thus, in many cases, reconfigurable hardware devices provide a good compromise between software and hardware solutions [47, 62]. The resulting reconfigurable systems enable optimization across a broad range of engineering applications and possess the ability to react to dynamic mission requirements.

Field programmable gate array (FPGA) is a reconfigurable device which can be used to implement many applications, such as a real-time traditional adaptive FIR filters, facial image associative memory or other intended applications. The implementation can potentially be a full on-chip solution where the adaptation or self-configuration is performed by a user defined configuration logic controller in Figure 2-1. In this work, we focus on the software simulation of the neural net processor to demonstrate the potential of our approach. Figure 2-1 also shows an embedded memory space which can store a set of different
circuit configurations.

![Diagram of reconfigurable system model]

Figure 2-1: Reconfigurable system model

The reconfigurable system platform is designed based on cellular and polymorphic architecture. A cellular architecture [3, 63] is a type of computer system architecture prominent in parallel computing with multi core architecture design. A cellular architecture gives the programmer the ability to run large numbers of concurrent threads within a single processor. Each cell is a compute node containing thread units, memory, and communication. Speed-up is achieved by exploiting thread-level parallelism inherent in many applications. Polymorphic is defined as having, taking, or passing through many different forms or stages. Polymorphic architectures [64, 65] can reconfigure and adapt to different requirements, which can support multiple sensors, computation objectives and actuator modules.
2.2.2 Retargetable Information Processing Applications

Historically, applications including software and robots, for industrial purposes involved little or no learning. Recently, a growing interest in unstructured environments has encouraged learning intensive design methodologies. The emerging class of applications must be able to interact responsively with people and other applications providing assistance and service that affect everyday life. Applications may learn by adjusting parameters, exploiting patterns, evolving rule sets, generating entire behaviors, devising new strategies, predicting environmental changes, recognizing the strategies of opponents or exchanging knowledge with other applications.

A retargetable information processing application is a software/hardware cooperative design that can translate application objectives or missions into computational instructions and data structures to run on a common reconfigurable processing system. The computational instructions and data structures produced by the retargetable information processing application, shown in Figure 2-2, usually has limited ability compared to those produced by an application specific computing method. Typically, the design of a retargetable information processing application separates the computational instruction generation from data structure representation. It uses the cellular architecture in the computational instruction generation and the polymorphism for data structure representation. Thus, the computations for any intended application can be generated for one common reconfigurable processor system. The optimization of the retargetable
information processing application does not require detailed, specific knowledge of the reconfigurable system architecture platform and how the computational instructions are executed. The application optimizations are usually done by a reconfigurable system platform compiler which is outside the scope of this investigation.

The reconfigurable system has high potential for supporting the retargetable information processing applications. Cellular polymorphic architecture introduces an approach to implement evolutionary systems that can support multiple sensor and retargetable computation objectives. Cellular polymorphic architecture provide the computing foundation for retargetable information processing applications by establishing multi layer network (neural net like) that can reconfigure processing cellular architecture and architectural elements to optimize performance for changing objectives, sensor configurations, and operational constraints during a mission or over the life of the system. Figure 2-3 shows the cellular architecture of neural net and the polymorphic architecture of FPGA.
2.2.3 Hardware/Software System Configurability

Hardware/software system configurability is categorized as follows: static (non)-configurable, dynamic-configurable, self-configurable, and evolvable-configurable system. Static-configurable system or non-configurable system is a system with the total deficit of configurability. The static-configurable system cannot be modified. For example, a sea shell is considered as a static-configurable system because the shell cannot changes its structure or functionality without damaging itself. There raise an issue that the shell can be used as a decoration, a skipping rock, or a bullet, which changes its functionality from a furniture to a toy or a weapon. There is no clear cut in this example because no object that has only one functionality or purpose. If there exists such a system or an object which has only one function, it will be well suited for static-configurable system.
Dynamic-configurable system is defined as a system that its function is dependent on the environment stimulus, an external controller, or a human operator. The dynamic-configurable system can change its structure and functionality upon the external stimulus. An example of dynamic-configurable system with human operator interaction is a common, simple reading lamp. The reading lamp can be turned on or off depending on the reader. Another kind of lamp such as a smart lamp can adjust its illuminant to keep the room light intensity at a pre-defined level. The smart floor lamp is usually equipped with a photo sensitive sensor which monitors the light intensity of the room. The smart floor lamp operates upon the room light stimulation to change its illumination.

Self-configurable system solves the issue of requiring a human operator interaction. The system has an ability to change its structure and functionality as suggested in its configuration managing policy depending on the external stimulus to the system from the environment to maintain its behavior. Moreover, the self-configurable system is automated such that it can modify its structure based on the system configuration managing policy. A self-configurable system can receive a new set of configuration managing policies from a human operator; however, it may never modify the given policy on its own. This kind of system is more sophisticated than the dynamic-configurable system in a way that it is not restricted to the preset initial system structure. An example of self-configurable system can be seen in laboratory setups or commercial product back-ends, such as smart network routing. A smart network routing [66] goal is to reduce network congestion which can be done with network traffic protocol policies. Other
examples are [67, 68, 69].

The last system type is evolvable-configurable system. With the current technology and research, it is the most complex system of all. Evolvable-configurable system has all properties of self-configurable system plus additional properties. Evolvable-configurable system has an ability to modify its configuration managing policy toward a given goal. The process of modifying the configuration managing policy is totally automated based on some priori knowledge. By putting the goal seeking feature into the system specification, this evolvable-configurable system stands out for potentially many applications especially for evolutionary system. For example, a biological cell system has a goal to survive in the environment. It grows and mutates upon the environment stimulus; and, at the same time, it maintains its survival rate. With these definitions, the biological cell can be categorized into an evolvable-configurable system.

The purpose of this work concentrates mainly on the study of self-configurable and evolvable-configurable systems, which will refer to, in general, as evolutionary systems. The study looks at how the evolutionary system can be implemented for a few selected potential applications, specifically for retargetable information processing applications.
2.3 Information Cognitive Modeling

The term cognitive model applies to the study of human cognition particularly as it affects learning and behavior. It has two meanings in psychology and in computer science. Contemporary cognitive theory has followed one of two broad approaches: the developmental approach and the information processing approach. The developmental approach \[70, 71, 72\] is concerned with “representational thought” and the construction of mental models (“scheme”) of the world. On the other hand, the information processing approach \[24, 25, 26\] views the human mind as analogous to a sophisticated computer system.

2.3.1 Cognitive Models

In cognitive science, a cognitive model is a model of cognitive processing. It is the use of computers to model cognitive behavior (and sometimes the study of cognitive behavior to improve the usage of computers). Cognitive models are used to study intelligent or social behavior and emergent properties of a connectionist architecture \[73, 74\]. Cognitive modeling can also be understood as reverse engineering some aspect of human cognition by means of artificial intelligence \[41, 21, 29\] and machine learning \[75, 76\] in order to better understand these aspects.

Some of the most popular frameworks for cognitive modeling include good-old-fashioned artificial intelligence (GOFAI) \[41, 21, 29\], ACT-R \[77, 78, 79, 80, 81\],
and SOAR [82, 83, 81].

1. **Good-old-fashioned artificial intelligence (GOFAI)** is commonly used to denote a branch of artificial intelligence which mainly deals with symbolic problems. It is based on the assumption that thinking is nothing but symbol manipulation. Thus, it holds out the hope that computers will not merely simulate intelligence, but actually achieve it. In current artificial intelligence research [41, 21, 29], the term is often extended to GOFAIR (good-old-fashioned artificial intelligence and robot) to reflect the importance of symbolic reasoning in traditional robotics. Often the term is used to distinguish systems that do not employ connectionist or statistical machine learning algorithms which have come to play a major role in artificial intelligence, robotics, and computer vision since late-1990s.

2. **ACT-R** [77, 78, 79, 80, 81] (Adaptive control of throughRational) is a cognitive architecture aiming to define the basic and irreducible cognitive and perceptual operations that enable the human mind. In theory, each task that human perform should consist of a series of these discrete operations. Most of the ACT-R basic assumptions are also inspired by he progresses of cognitive neuroscience and, in fact, ACT-R can be seen and described as a way of specifying how the brain itself is organized in a way that enables individual processing modules to produce cognition. Like other influential cognitive architectures, the ACT-R theory has a computational implementation as an interpreter of a special coding language. This enables researchers to
specify models of human cognition in the form of a script in the ACT-R language. The model assumptions are based on numerous facts derived from experiments in cognitive psychology and brain imaging. [84, 85]

3. **SOAR** [82, 83, 81] is a symbolic cognitive architecture which is both a view of what cognition is and an implementation of that view through a computer programming architecture for artificial intelligence. It has been widely used by AI researchers to model different aspects of human behavior. The main goal of the SOAR project is to be able to handle the full range of capabilities of an intelligent agent, from highly routine to extremely difficult open-ended problem. In order for that to happen, according to the view underlying SOAR, it needs to be able to create representations and use appropriate forms of knowledge. Also underlying the SOAR architecture is the view that a symbolic system is necessary and sufficient for general intelligence.

### 2.3.2 Machine Learning

Machine learning [5, 29, 86, 87] is characterized by formalism and statistical analysis. Machine learning is concerned with the development of algorithms and techniques that allow computers to learn. Machine learning overlaps heavily with statistics. Many machine learning algorithms have been found to have direct counterparts with statistics.

Increasingly, researchers have adopted hybrid learning strategies which attempt to blend learning methods once viewed as distinct. In this project, we
employ a few machine learning techniques to train the evolutionary system. Despite the fact that machine learning technique boundaries can be blurred in places, we will adopt the following five classifications:

1. **Artificial Neural Networks** is a supervised, learning-with-a-trainer approach where knowledge is learned by adjusting weights between nodes of a neural net. Artificial neural networks (neural nets) [88] are algorithms based very loosely on the neural phenomenon of spreading activation. Guided by reinforcement given during training episodes, neural nets can encode knowledge and skill implicitly as associative connections between nodes. Stimulation introduced to the input nodes of a neural net travels between layers of the network to produce some output on the other end. This output is evaluated by a trainer who applies reinforcement to alter the weights of synaptic connections and thereby change the way the network will respond. In this way, neural nets allow human knowledge and guidance to orchestrate the learning process. Such techniques, where high-level reinforcement is applied by a knowledgeable teacher, are often referred to as supervised learning. [89]

2. **Evolutionary Learning** is an unsupervised, learning-with-a-critic approach where controllers are derived deductively by alterations to an initial population of program code. Evolutionary computing [90] is a term used to indicate a group of unsupervised learning methods where behavior is evolved deductively from a randomly generated or seeded population. Evolutionary
methods draw their inspiration from natural processes of biological evolution. Evolutionary computing includes a variety of computational systems including genetic algorithms, evolutionary strategies, learning classifier systems, and genetic programming. Evolutionary computing [5, 6] adopts a population-based, generational approach to search, exploration, and optimization. The evolutionary computing utilizes crossover, mutation and fitness proportionate reproduction operators. [23, 27, 5, 90]

3. **Search and Heuristic** is a knowledge based system that contains some of the subject-specific knowledge of one or more human experts. Various search techniques in machine learning processes require some priori knowledge relating to the search environment and gradient information is needed. Such techniques initially populate a variable space with trial solutions and the extent of subsequent search and heuristic from initial points depends upon their relative performance. Further sampling of diverse regions of that space can continue throughout the search process through the action of optimization operators. The techniques can identify multiple high-performance solutions from complex variable spaces and multi-objective techniques such as Pareto optimality can also be integrated to provide a set of good solutions for further offline evaluation. Heuristics can be introduced to generate or limit the solution set and to improve solution average performance. Heuristics may relate to maintaining a diverse solution set as opposed to a set comprising of the best individuals. Premature convergence can be avoided
and exploration approach can be introduced with heuristic into the search process. [91, 92, 93]

4. **Reinforcement Learning** is an unsupervised, learning-with-a-critic approach where mappings from percepts to actions are learned inductively through trial and error. An evolutionary system can also learn using an automated critic that guides development by reinforcing or “punishing” actions (or intended actions). Reinforcement learning [94, 86] is referred specifically to an unsupervised, learning-by-critic approach where each state of the environment stores a value used to reward or to punish the evolutionary system’s progress. The goal of reinforcement learning is simply to devise some mapping from perceptual states to actions that will maximize the total reward. Reinforcement learning suggests an evolutionary system to explore an environment in which it considers its current state and takes actions. The environment, in return, provides a reward which can be positive or negative. Reinforcement learning attempts to find a policy for maximizing aggregate reward for the agent over the course of the problem. The environment is typically formulated as a finite-state Markov decision process [95, 86, 96], and reinforcement learning algorithms for this context are highly related to dynamic programming techniques. State transition probabilities and reward probabilities in the Markov decision process are typically stochastic but stationary over the course of the problem. [97, 21, 98, 99, 100, 22]
5. *Learning by Imitation* is a design methodology which uses a biologically inspired developmental paradigm to enable learning by emulation. Imitative Learning is more an approach than a specific computational means. Theoretically, it might involve any of the means described above. Here it is characterized separately because it represents a unique methodology based on the notion that robots can be shown rather than told how to behave. Eventually, a human should be able to demonstrate actions and supply auditory and visual cues to help the robot correctly perceive the instructions. A useful application is robots that can take instructions from a military commander. The robot would be able to respond to natural gestures and verbal instructions to provide services that cannot be otherwise be rendered without endangering human life. These services might involve delivering correspondences or scouting dangerous areas. Unfortunately, even a seemingly simple task such as filling a gas tank is far from trivial for a robot trying to learn through emulation. A robot equipped with precision actuators allowing it to reach, grip and manipulate objects is still faced with the need to decompose the task into: removing the cap, inserting the nozzle, squeezing the handle, waiting for the tank to fill, removing the nozzle, etc. Only after it has identified these subtasks can it map them to low-level behaviors it already knows such as reaching and grasping.
2.3.3 Multi-Sensory Information

Humans and animals have evolved the capability to use multiple senses to improve their ability to survive. For example, it may not be possible to assess the quality of an edible substance based solely on the sense of vision or touch, but evaluation of edibility may be achieved using a combination of sight, touch, smell, and taste. Similarly, while one is unable to see around corners or through vegetation, the sense of hearing can provide advanced warning of impending dangers. Thus multisensory data fusion is naturally performed by animals and humans to achieve more accurate assessment of the surrounding environment and identification of threats, thereby improving their chances of survival.
Chapter 3

Cognitive Processing on Evolutionary Platform

Evolutionary platform uses a collection of artificial intelligence and machine learning, such as evolutionary techniques and adaptive algorithms, to provide self-configurability. The methodology is based on underlying dynamic reconfigurable hardware platform. They address advanced applications in challenging environments such as on-board imaging and identification, remote sensing and processing, and autonomous UAVs and robots.

3.1 Cognitive Processing Overview

The main innovation of our method is a cognitive processing scheme consisting of three phases:

1. multisensory signal filtering into features,
2. fusion of the sensory features into a signature associated with a target, and

3. cognitive matching of the input sensory signature to similar target signatures pre-stored in the memory or in the knowledge base.

The key element of our method is the use of novel evolutionary platform approach for offline learning and training (evolutionary learning) of realistic target signatures under several experimental scenarios.

A key issue of our approach is how to analyze and evaluate multiple sensory data concerning the target in real time. Patterns of diverse sensory data could be weighted to provide real time responses. Sensory patterns could be recognized by adaptive techniques and systems that have been trained, offline, based on data from previous experimental scenarios. A key property of the system is the adaptation, training and learning capability of the system based on a cognitive processing scheme to provide responses in real time.

3.1.1 Target Profiles and Signatures

By target we mean an object or closely related objects that are of high interest to the system objective for the purpose of tracking, identification and assessment. A target profile is a collection of multisensory information, i.e. images, sounds, data, etc., associated with a target. Multiple profiles are usually needed for a particular target type. For example, an M60 series tank may have many infrared vision images, a few engine sound samples and an engine oil chemical emission. The profiles are used to generate a single signature vector associated with the
target. We assume that the system setup has been properly built to store \( n \) target profiles and signatures in the memory or knowledge base.

### 3.1.2 Cognitive Processing Platform Architecture

Our proposed cognitive processing is based on evolutionary system architecture, shown in Figure 3-4. We use two-level cognitive processing to handle the target cognition:

1. sensory processing on raw signals from the sensors;

2. cognitive processing based on elaborated feature data.

To perform both processes, we propose an evolutionary platform approach for learning and training technique (evolutionary learning) using training sets containing both raw and elaborated features. This technique is implemented by multisensory neural net processing, to be described shortly.

### 3.1.3 Technical Approach

Our cognitive processing consists of three phases shown in Figure 3-5.

1. Sensory pre-processing

2. Interfusion into signatures

The cognitive processing can operate in two distinct modes, *normal* operation and *training* mode. In normal operation, the cognitive processing goes through the three phases in real time to process and evaluate the sensory data and respond to inquiries. In training mode, the cognitive processing employs an evolutionary platform approach for learning/training technique (evolutionary learning); specifically, the above three phases have to go through evolutionary learning.

All three phases involve multiple neural nets in each phase forming a hierarchical structure comprising an ensemble of neural nets [101]. We will denote the evaluation of a neural net structure given by an input vector \( x \) and a set of net weights \( w \) as a function \( \mathcal{N}(x, w) \).

Evolutionary platform approach for learning/training (evolutionary learning) employs a neural net-based training and mapping approach between input
Figure 3-5: Cognitive processing (Classification/Identification)

and output representations via an evolutionary learning algorithm. Evolutionary learning supports offline training during traditional training sessions. It may also support on-line training during real time application. Evolutionary learning augments the three-phase cognitive processing mentioned earlier, Figure 3-5, i.e. sensory processing, signature fusion and cognitive recall.

### 3.2 Sensory Processing (Phase One)

Figure 3-6 shows multiple sensory raw data transformations through distinct processing filtering modules. The cognitive processing platform may be equipped with many heterogeneous filtering modules corresponding to distinct signal data types (sound, visual, infrared, etc). High level features of information are extracted from raw sensory data using appropriate transformation and filtering
techniques, e.g. Fourier or wavelets. However, other information can be preferential rather than traditional mathematically based. This is achieved with evolutionary preferential neural process learning.

The sensory processing phase consists of two parts:

1. the sensory pre-processing filters,

2. the feature extractors.

Figure 3-6 shows multiple sensory raw data transformations through distinct processing filtering modules. The evolutionary system architecture may be equipped with many heterogeneous filtering modules corresponding to distinct signal data types (sound, visual, infrared, etc).

Figure 3-6: Sensory processing with pre-processors and feature extractors

High level features of information \((a'_i)\) can be extracted from raw sensory data \((a_i)\) using appropriate transformation and filtering techniques \((P_i)\), e.g. Fourier
or wavelets. Thus, $a'_i = P_i(a_i)$ where $i$ indicates the $i$-th sensory pre-processing filter. However, other information can be preferential rather than traditional mathematically based. In our approach, the feature extractors use a collection of neural nets such that each neural net is responsible for one feature extractor operation, in similar way as described in [102, 103]. A feature extractor accentuates the pre-processed raw data and projects it into potential features based on its experience from training data that have been subjected to different sensory information. Since the evolutionary system architecture incorporates $s$ sensors, the sensory processing phase needs $s$ feature extraction channels which implies $s$ neural nets, as shown in Figure 3-6. Specifically, the output $f_i$ of the $i$-th feature extractor (corresponding to the $i$-th feature extraction channel or $i$-th sensor) is a function of both the pre-processed information (vector) ($a'_i$) and a set of the net weights ($w_i$) and net structure ($N_i$). This can be expressed by $f_i = N_i(a'_i, w_i)$ where $i = 1, \ldots, s$. The output $f_i$ is a feature vector of length $r$ representing the sensory reading of the potential target features.

A note about pre-processors and feature extractors should be made in reference to Figure 3-6. Pre-processing is sensor specific such as audio filtering and image filtering extracting only the necessary part of data to follow up by feature extraction. Some sensory information maybe used to extract (reduce or compress) more than one target features such as low, mid, or high-range frequencies, with each frequency designated to a specific feature extraction channel. The result of pre-processing is usually in vector form and is mapped into potential target feature vectors using the feature extractor. For example, the system receives a
thermal image from an infrared vision sensor. The corresponding pre-processor extracts the thermal image shape and represents it in a vector format [102, 104]. Then, the feature extractor maps the shape representation vector into a feature vector for further analysis in the fusion recall process.

3.3 Signature Fusion (Phase Two)

This phase is illustrated in Figure 3-7. The essence of this phase is to transform the extracted high level information from the sensory processors and merge it into a single vector stream. Merging uses statistical parameters such as mean, weighted average, etc. However, merging can be biased with existing knowledge rather than using standard unbiased parameters. This bias can be achieved using evolutionary neural process learning.

The signature fusion phase merges all the extracted feature vectors from the sensory processing phase (Phase 1) into a single vector using a collection of feature interfusors. A feature interfusor is a neural net which is responsible for consolidating a set of values into a single number, in similar way as discussed in [1, 105]. All feature vectors have a length of $r$ (defined by the evolutionary system architecture). Every feature interfusor combines all assigned vector elements from the feature vectors and calculates their weighted average. This is partially biased from its experience governed by training data subjected to its assigned vector element. Since the feature vectors all have length $r$, the fusion recall phase needs $r$ feature interfusors which implies $r$ neural nets, as shown in Figure 3-7.
The output $g_i$ of the $i$-th feature interfusor is a function of a vector which collects the $i$-th element from all extracted feature vectors ($b_i = (f_{1i}, \ldots, f_{si})$) and a set of the net weights ($w_i$) and net structure ($\mathcal{N}_i$). Hence, we can represent this as, $g_i = \mathcal{N}_i(b_i, w_i)$ where $i = 1, \ldots, r$. We consider the output vector $\hat{g} = \{g_1, \ldots, g_r\}$ as being a signature vector of length $r$ representing the potential recall (interfused) target signature according to the sensory information.

![Figure 3-7: Signature fusion with feature interfusors](image)

The vector stream produced by merging can be viewed as a snapshot signature concerning a particular target. Note that a target may be associated with many signatures which may form signature profiles.

In recent years, multi-sensor data fusion has received significant attention for both military and nonmilitary applications. Data fusion techniques combine data from multiple sensors, and related information from associated databases, to achieve improved accuracies and more specific inferences than could be achieved by the use of a single sensor alone [106, 107, 108]. The concept of multi-sensor data fusion is hardly new. The integration of information from multiple sensors
has been an active research area in recent years and well documented in great detail [109].

The emergence of new sensors, advanced processing techniques, and improved processing hardware make real-time fusion of data increasingly possible [110, 111, 112]. Recent advances in computing and sensing have provided the ability to emulate, in hardware and software, the natural data fusion capabilities of humans and animals. Currently, data fusion systems are used extensively for target tracking, automated identification of targets, and limited automated reasoning applications. Data fusion technology has rapidly advanced to an emerging true engineering discipline with standardized terminology (see Table 3-1), collections of robust mathematical techniques [106, 107, 108, 113], and established system design principles, the minimum variance estimation [114, 115], the maximum likelihood technique [116], and the linearly constrained least squares neural fusion approach [117]. The algorithm of interacting multiple models was considered in [118]. The asynchronous data fusion method was studied in [119]. The sub-optimal approach using the information matrix filter is in [120].

Applications for multi-sensor data fusion are widespread. Military applications include: automated target recognition (e.g., for smart weapons), guidance for autonomous vehicles, remote sensing, battlefield surveillance, and automated threat recognition systems, such as identification-friend-foe-neutral (IFFN)
Fusion | The integration of information from multiple sources to produce specific and comprehensive unified data about entity.
---|---
Alignment | Processing of sensor measurements to achieve a common time base and a common spatial reference.
Association | A process by which the closeness of sensor measurements is completed.
Correlation | A decision-making process which employs an association technique as a basis for allocating sensor measurements to the fixed or tracked location of an entity.
Correlator-Tracker | A process which generally employs both correlation and fusion component processes to transform sensor measurements into updated states and covariance for entity tracks.
Classification | A process by which some level of identity of an entity is established, either as a member of a class, a type within a class, or a specific unit within a type.
Situation Assessment | A process by which the distributions of fixed and traced entities are associated with environmental, doctrinal, and performance data.
Threat Assessment | A structured multi-perspective assessment of the distributions of fixed and tracked entities which result in estimates of (e.g.):
- expected courses of action
- enemy lethality
- unit compositions and deployment
- functional networks (e.g. supply, communications)
- environmental effects

Table 3-1: Standardized terminologies in data fusion technologies from [1]
The basic goal is to improve the performance and robustness of the system by properly combining the information obtained from various decentralized sensors while minimizing the communication traffic. Multi-sensor data fusion is done to produce a model or representation of a system from a set of independent data sources, from which a single view or perception of some external environment or system is found or detected. Multi-sensors data fusion is the estimation of the information of an observed object based on remote measurements taken from one or more sensors at fixed locations or on moving platforms. Therefore, data fusion is the continuous process of assembling a model of the domain of the interest utilizing data from disparate sources [109].

For many applications involving real time, the desired domain model is the state vector of a dynamical process, such as an observed airborne vehicle. The combination of the information from the sensors and subsequent estimation of the state of the environment should be done in a consistent, coherent manner such that the uncertainty is reduced. Unfortunately, every sensor, event the best and expensive ones, has limited accuracy and reliability. It can have an uncertainty and sometimes, under some conditions, it does not work correctly. Hence, fusion systems may receive imprecise, incomplete, and inconsistent information about their parameters of interest. Multi-sensor data fusion techniques, which combine data from several independent sensors, have emerged as methods to
overcome these problems. A system can have accurate information by combining readings from several redundant sensors or different sensors. In addition, a system that uses several independent sensors will be less sensitive to sensor failure than a system with a single sensor alone.

3.4 Cognitive Recall (Phase Three)

This phase is illustrated in Figure 3-8. The cognitive recall phase consists of two key parts: the associator pipes and the associative memory. The latter contains all target signatures and profiles known to the system in its knowledge base or database. The interfused signature from the signature fusion phase (Phase 2) is compared against all stored known target signatures; this is done by a collection of signature associators. The role of the associators is to measure the statistical similarity (or difference) between the merged information or interfused signatures (from Phase 2) and the stored target signatures in the memory or knowledge base regarding the target. The similarity measures determine the association matching levels between the interfused signatures and stored target signatures. There are several similarity measures based on distance metrics such as the Euclidean, the root-mean square (RMS) and so on. However, often in practice, similarity measures in real association are not mathematically well defined. We address this issue using an evolutionary neural process learning to handle undefined measurement.

In our approach, a signature associator is a neural net which is in charge of
comparing two signatures and producing a confidence level of their match. The comparison (association) between the two signatures of an associator is based on its experience governed by different training data that were subjected to its corresponding target. Since the system stores $n$ target profiles and signatures, the interfused signature may match any one of the $n$ stored signatures. The cognitive profile recall phase needs $n$ signature associators to support the $n$ comparisons which imply $n$ neural nets, as shown in Figure 3-8. The database box is information storage for the target signatures and profiles generated during the system setup. The output $h_i$ of the $i$-th signature associator is a function of the recall signature vector ($\hat{g}$) and a set of the net weights ($w_i$) and net structure ($N_i$).

Hence, we can express this by $h_i = N_i(\hat{g}, w_i)$ where $i = 1, \ldots, n$. The output vector $\hat{h} = \{h_1, \ldots, h_n\}$ is a confidence vector of length $n$ representing the input target cognition confidence levels of the associations between the input target and all known targets. The target with the highest confidence level is identified as the potential target. Each stored target ($i$) is assigned an identification vector ($Q_i$) of length $n$ which represents a unique target with a high identification value (confidence level) at the $i$-th element and low values elsewhere. The output vector ($\hat{h}$) is compared with the target identification vector ($Q_i$) to determine the system performance (cognition error).

We assume that a knowledge base about target signatures and profiles should be prepared based on data from previous experimental scenario. This task is quite involved since it deals with collecting and processing large amounts of real data. This knowledge base may be quite large and should be securely stored.
3.5 **Evolutionary Learning (Training Scheme)**

To say that an evolutionary system can adapt actually communicates very little about the system control strategy. On one extreme, applications may merely fine-tune already hard-coded behavior. On the other, there are applications that write their own programs from scratch using a randomly generated pool of binary numbers. Still others may exchange programs or portions of programs with other applications to produce co-evolutionary adaptation or learning. Although there are countless variations, the most interesting adaptation or learning occurs when the evolutionary system can devise its own approach from the bottom up. Some evolutionary systems are even able to adapt or learn the learning strategy for a given task.

An evolutionary adaptation scheme (learning scheme) is a collection of collaborative artificial intelligence modules, machine learning, cellular polymorphic platform, and configuration policy managers under the control of an evolutionary service that presents a common managing policy to the system, shown in Figure 3-9. The evolutionary adaptation scheme definition requires a control
by an evolutionary service with independent interactions to artificial intelligence and machine learning on cellular polymorphic platform that conform to a single, clearly defined configuration managing policy.

Artificial intelligence [6, 21, 29, 25] and machine learning [19, 6, 20, 21, 22] is a research area focus in intelligent behavior, learning, and adaptation in machines. Research in artificial intelligence is concerned with producing machine to automate tasks requiring intelligent behavior. Examples include control, planning and scheduling, the ability to answer diagnostic and consumer questions, handwriting, speech, and facial recognition. It has become an engineering discipline, focused on providing solutions to read life problem, software applications, traditional strategy games like computer chess and other video games.

One of the first domains in artificial intelligence research has been computer chess. This is not surprising, if we consider the possibilities people thought computers would have in the near future. Using these possibilities, people thought
it would be possible to let the computer perform tasks for which humans need intelligence, whatever that may be.

### 3.5.1 Preparation

The goal of evolutionary platform approach for learning/training (evolutionary learning) on neural net is to incorporate

1. adaptive training on neural net coefficients,

2. evolutionary modification on neural net structure, and

3. enriched training with varying training data.

Though, the classical neural net training is capable of adaptive coefficients, it suffers from an inflexible structure of the network and a limited and fixed training data. The evolutionary learning produces a well-trained neural net (coefficients and structure) regardless of initial neural net and initial training data.

The evolutionary learning for neural net is a three-hierarchical-iterative-loop process. It takes an arbitrary initial neural net (coefficients and structure) and initial training data as inputs to produce a well-trained neural net (coefficients and structure) with respect to minimal energy error\(^4\), structural evolution decisions\(^9\), and available training data\(^1\). The three iterative loops are nested in the following order (from most inner loop to most outer loop): coefficient training, structural evolution, and enrichment process, as shown in Figure 3-10. The following sections describe the details of each loop in the order.
The proposed evolutionary learning (training scheme) differs from the traditional training \cite{16, 49} in an evolutionary aspect, that is, the growth/shrinkage of the neural net structure and the training data themselves progressively evolve. Thus the key advantage of our evolutionary learning scheme is based on two processes, the neural net structural evolution and the training data enrichment, which progressively iterate to produce enhanced training.

As discussed earlier, all three phases of the cognitive processing employ a set of neural nets. Initially, all neural nets are initialized with different sets of net weights \((w)\) and net structures \((N)\). This means a fully connected neural net with specific numbers of hidden layers and hidden neurons including their connection weights. The net weights and net structure are determined by prior knowledge about the net input-output training data derived from the system input-output training information. The net structure is evaluated with a vector input \((x)\) and produces another vector output \((y)\). Each neural net must undergo a training process \((T)\) with a set of training data \((T)\). One complete training process resulting in a modified neural net is considered a training iteration. Hence, we represent this by \(N = T(N(x, w), T)\) where \(x \in \{I_k | (I_k, O_k) \in T\}\).

The training scheme consists of three nested processes:

1. coefficient training process (inner loop),

2. structural evolution process (middle loop), and

3. training data enrichment process (outer loop),
as illustrated in Figure 3-10. Each neural net in all three phases must undergo this hierarchical three-nested-loop training process. This includes all $s$ nets in feature extractors, $r$ nets in feature interfusors, and $n$ nets in signature associators shown in Figure 3-6, Figure 3-7, and Figure 3-8. Note the overall training process can train multiple neural nets in parallel.

![Evolutionary learning (training scheme)](image)

**Figure 3-10: Evolutionary learning (training scheme)**

### 3.5.2 Nested-Loop Training

Each neural net is initialized with a set of net weights and a net structure based on priori knowledge about the net input-output training data derived from the system training information. The coefficient training process (inner loop) is a variation of the back propagation algorithm [16, 49] using adaptive learning rates as training parameters to accelerate the convergence of the net weights, as discussed in Subsection 2.1.2. One learning rate coefficient is associated with one net weight. The learning rates control how fast the net weight estimations are modified. All learning rates are initialized to zero and updated with a function of net output energy error in every coefficient training iteration.
The structural evolution process (middle loop) is a significant enhancement to the classical neural net learning such that the net structure is modified and evolves to improve the output accuracy and help the weights to converge faster. Lastly, the training data enrichment process (outer loop) is responsible for supplying training data to the other two training processes (inner loop and middle loop) in a way that the neural net may explore the input-output training data space efficiently. This helps to further improve the output accuracy and convergence.

### 3.5.3 Definitions and Notations

The following describes the hierarchical three-nested-loop training process, shown in Figure 3-10. This is applicable to all neural nets used in the three phases of the proposed cognitive model. The diamond box in each loop represents some terminal conditions for the loop, respectively. Initially, the net is initialized with a set of weights \( w \) and a network structure \( \mathcal{N} \) based on priori knowledge derived from the system input-output training information. The training set may consist of \( d \) net input \( (I_k) \)-output \( (O_k) \) data pairs. Hence, the training set \( \mathcal{T} \) can be represented as \( \mathcal{T} = \{(I_k, O_k) | k = 1, \ldots, d\} \).

The input-output training data of each neural net may be different. We will denote a generic net input as \( x \) and its generic output as \( y \). Therefore, we can represent one net evaluation as \( y = \mathcal{N}(x, w) \). In general, the net training process
is to update the current net weights in the inner loop, Figure 3-10, and net structure in the middle loop, with some training data ($\tau \subseteq T$). Hence, we can represent the net training by $\mathcal{N} = T(\mathcal{N}(x, w), T)$.

### 3.6 Coefficient Training (Inner Loop)

The neural net training starts from the coefficient training process (inner loop) which is responsible for net weights adjustments, shown in Figure 3-11. With the initial neural net and initial training data, the coefficient training process trains the neural net with the given training data for a number of training iterations$^{11}$. As an example, the coefficient training may predefined one successful coefficient training loop as 10 training iterations or one coefficient training milestone. During the coefficient training process, training progress records and collects several pieces of information for later use, such as energy error, last input tap coefficients, last feedback tap coefficients, and neuron coefficient changes. The coefficient training ($T_c$) updates the net weights ($w$) to minimized the energy error$^4$ between the net output ($y$) and the training target ($O_k$) by neural net back propagation. Hence, we can write the update as $w = T_c(\mathcal{N}(x, w), \tau)$. The coefficient training repeats this process in a predefined number of iterations (denoted as one coefficient training milestone). Once the coefficient training completes the milestone, the coefficient-trained neural net suspends the inner loop and moves on to the structural evolution process.
3.7 Structural Evolution (Middle Loop)

In this section, we will discuss the framework that evolution (growth and decay) is typically used for many (hardware/software) application design problems. The majority of published evolution works, both applications and theoretical analysis, refers to optimization problems which can be seen as search problems in some high-dimensional search space of usually large size (finite or infinite). The components to be optimized could be parameters which need to be set at appropriate values (continuous or discrete). Many optimization problems have the well-defined finite dimensionality of the search space; some do not.

Evolutionary (theory) computation techniques [23, 5] provide robust method in searching for an optimum in a finite-dimensional search space. Based on the Darwinian theory of natural selection, they attempt to obtain the best solution...
by carrying out global optimization. They use suitable coding (genotype) to represent possible solutions for a problem and guide the search by using genetic operators and the principle of “survival of the fit-test”. Of these algorithms, genetic algorithms [5, 16] have established themselves as powerful search and optimization tools in problem solving and function optimization [5]. The optimization problem has typically been seen as starting with a population of random points effectively spanning and coarsely sampling the whole search space. Successive rounds of selection, reproduction and mutation are intended to focus the population of sample points towards fitter regions of the space, homing in on an optimum or near-optimal region.

However, genetic algorithms cannot be used to evolve complex models. Some problems—including most hardware design problems, such as self-configurable adaptive filters, equations—do not have a finite-dimensional search space. If there is no predetermined number of components to be used in the design, then the standard evolutionary theory will not apply. An alternative approach, which one might be uncertain initially how many components might be needed, is incremental/decremental iterative evolution. The iterative evolution uses a single point from possible population which a sequence of increasingly complex tasks is required for more complex design or the number of components in a structure is to be reduced. If there is no pre-defined range for the number of components in a structure to be designed, then design genotypes will be variable in length. The genotype is potentially unbounded in length; thus, the search space is unbounded. It is impossible for an initial random population of finite size to
effectively sample from all parts of the unbounded space.

Some beneficial solutions have been suggested, such as genetic programming [20] has been developed to overcome this limitation. Genetic programming can perform symbolic regression [20]. It is advantageous in that a priori assumptions and prior knowledge about the structure and size of the solution are not necessary. Moreover, this technique can automatically select the system inputs to yield a model structure that accurately fits the input-output response of the system and provides a descriptive solution. Due to its advantages, genetic programming has been successfully used in the study of engineering problems, such as process modeling, auto control, signal processing, hydrological modeling, ecological modeling, pattern recognition, and data mining [126, 127, 128].

3.7.1 Structural Evolution Growth and Decay

Evolutionary service takes on the major role in evolutionary system development. The system evolution can be considered either in incrementally growth or diminishing decay. The incrementally growth refers to the evolutionary system is expanding its structure to be able to achieve given operation objectives. On a contrary, the system may take a side-step or a step-back diminishing decay its structure to yield minimal design. The two development approaches of evolutionary system can be considered as growth and decay similar to the nature.

System growth is a change solely in the system structure. This means the component functionalities are to remain unaltered in growth; but, its structure
element can be added or expanded. An example of system growth can be seen in biological cell development. During a normal cell development, the cell components and their structures have been developed. The number of components might change due to the replication in the cell, but the functionalities of the components remain the same.

Another development in a system is diminishing decay which changes solely the structure of the system, while the system functionalities remain. System decay can be induced to find the minimal system structure design. Some decays lead to horrific result such as component malfunction and system failure. However, some decays yield great optimization results as seen in animal evolution.

The system growth and decay are under the influence of configuration policy management. The system development is also applicable to the configuration policy within evolving-configurable systems. The overall evolutionary service protocol is shown in Figure 3-12.

### 3.7.2 Structural Evolution Parameters

The structural evolution process (middle loop) is responsible for neural net structural adjustment. It updates the structure by adjusting four parameters (if applicable):

1. the number of filter input taps,

2. the number of filter feedback taps,
Figure 3-12: Overall evolutionary service protocol of system development

3. the number of hidden neurons, and

4. the number of hidden layers.

This process follows a set of rules that determine how to adjust these parameters. For example, a rule is dedicated to increment, decrement, and no-change decisions of the number of hidden neurons. The rule evaluates the energy error record from every iteration of the inner loop and suggests one of the three decisions (increment, decrement, or no-change) on the number of hidden neurons. With the coefficient-trained neural net and its training progress record from the coefficient training process, the structural evolution process evaluates the training progress record to decide if the net structure is appropriate, Figure 3-13. If the net structure is inappropriate, the structural evolution modifies the current net structure with respect to evaluated decisions. The structurally-evolve neural net must redo the coefficient training process.
The structural evolution ($T_s$) updates the net structure ($N$) to further minimize the energy error of the net and to help the net weights converge faster. This structural evolution is different from other related works such as in [129, 130, 131] which uses genetic algorithms to randomly explore the possible net structures. Although the genetic algorithm may produce an improvement, it cannot guarantee good results every time. Hence, we can represent the evolutionary update by $N = T_s(N(x, w), \tau)$. The structural evolution process evaluates these decision rules once every time the inner loop is finished. The process repeats by restarting from the inner loop in a pre-defined number of coefficient training milestones (denoted as one structural evolution milestone). Once the structural evolution training completes the milestone, the structurally-evolved neural net suspends the middle loop and moves on to the training data enrichment process.

\textit{2) Structural Evolution}

![Diagram](image)

Figure 3-13: Structural evolution
3.7.3 Structural Evolution Decisions

The following section describes the structural evolution decision mechanism. Structural evolution decision mechanism for net structure represents four adjustable structure parameters: input tap, feedback tap, hidden neuron, and hidden layer. Four separated neural nets are responsible for each structure parameter evolution decision, as illustrated in Figure 3-14 and Figure 3-15.

4 Evolution Decision

5 Input Tap Decision

Figure 3-14: Four separated neural nets for evolution decisions

Figure 3-15: Dedicated neural net for input tap parameter evolution decision
These parameter decisions approximate likelihood suggestions if the parameters should be increased, decreased, or no-changed. The structural evolution decisions evaluate information from the training progress record collected during every coefficient training process. Table 3-2 presents the list of information needed for each parameter evolution decision.

<table>
<thead>
<tr>
<th>Decision Network</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input tap</td>
<td>Energy error, Last input tap coefficient</td>
</tr>
<tr>
<td>Feedback tap</td>
<td>Energy error, Last feedback tap coefficient</td>
</tr>
<tr>
<td>Hidden neuron</td>
<td>Energy error, Hidden neuron change magnitude</td>
</tr>
<tr>
<td>Hidden layout</td>
<td>Energy error, Input neuron change magnitude</td>
</tr>
</tbody>
</table>

Table 3-2: Information needed for parameter evolution decisions

The decision neural net produces an output approximately between -1.0 and +1.0 representing the likelihood for increment the parameter if positive and decrement the parameter if negative. When the parameter evolution decision is +0.8, the decision likelihood is approximately 80% certain that the net structure should increase the corresponding parameter by one. All parameter likelihood suggestions are of the same significant level. After the parameter evolution decisions evaluate the likelihood, voting on the decision considers all the evaluated decision likelihood suggestions as equally weighted. The voting picks one decision with the strongest magnitude of suggestions (regardless of the signs) which represents the highest certainty of all decisions. The selected corresponding parameter with its structural evolution decision will determine the net structure modification only if the decision magnitude is above a pre-defined likelihood.
threshold. The structural evolution applies to only one parameter per structural evolution milestone. More than one parameter per modification is possible; however, selecting one parameter is for simplicity. The newly structurally-evolved neural net will be used in the next coefficient training session.

### 3.7.4 Structural Evolution Training

When standard genetic programming is used for model identification, the models generated may include a number of internal model parameters. These randomly generated parameters can have significant effects on the performance of a model, because their values affect the output of the model directly. As a potentially good model with favorable structure may be removed during the search, these internal model parameters have to be optimized in the model identification process. In many cases, one has difficulties in the optimization process because the parameters change dynamically with the evolution. Various methods have been proposed [126, 127, 128] to overcome the difficulties by adding the internal model parameters explicitly. These are somewhat artificially and not randomly generated.

A closer examination reveals that evolutionary computation techniques can be divided into two steps, namely, the structural evolution step and the parameter (coefficient) training step. First, genetic programming is employed to find and optimize the model structure in the structure evolution step. In this step, a model is generated along with its associated parameters (coefficients). In
the next step (the coefficient training step), the internal model coefficients are
optimized using the coefficient training process discussed in Subsection 2.1.2.
Therefore, the model structure will undergo the structural evolution process to
obtain its optimal coefficients. The structural evolution may be repeated until
the pre-defined terminal conditions are satisfied.

3.8 Training Data Enrichment (Outer Loop)

The training data enrichment process (outer loop) is responsible for training set
adjustments. Specifically, it updates the training data ($\tau$) by exploring and ex-
ploring the system input-output training data space. With the structurally-trained
neural net and its training progress record from structural evolution process, the
training data enrichment analyzes the current training data fitness condition. If
the current training data is unfitted (insufficient), the training data enrichment
requests more training data from the given training set ($\mathcal{T}$) and add them to the
current training data. Another rule, similar to the ones in the middle loop, eval-
uates the energy error record from every iteration in the inner loop and decides
if the training data is too clustered, too sparse, or just appropriate. If the training
data is clustered too closely in one region, some data will be removed. On the
other hand, if the training data is too sparse, more training samples will be re-
quested. Then the added training data and the current neural net redo the struc-
tural evolution process (including the coefficient training process), Figure 3-16.

The training data enrichment process ($\mathcal{T}_t$) manages training data, removes
training data, and requests additional training data for the two previous processes (inner and middle loop). The notion of training data enrichment is unique among existing related works such as [132, 133] which randomly generate subsets of data to train neural nets. Their approach may occasionally yield better neural net training; nonetheless, it is limited by the randomness of the training data subsets. Hence, we can represent the update as \( \tau = T_t(N(x, w), \tau) \). The current training data is evaluated every time the middle loop is finished. The process repeats by restarting from the structural evolution process (middle loop including the inner loop) in a predefined number of structural evolution milestones (denoted as one training data enrichment milestone).

The resulting neural net (coefficients and structure) is considered a well-trained evolutionary neural net with respect to structural evolution decisions and the available training data. Complete training data enrichment may take several structural evolution milestones, otherwise known as one training data enrichment milestone. Once the training data enrichment completes the milestone, it terminates the loop. The resulting neural net (weights and structure) is evaluated for performance metric measurement.

### 3.8.1 Adding/Removing Criterion

If the sufficient training data condition dissatisfies, the additional training data loop will request more training data from the given training set (\( T \)). The evolutionary training for neural net has no knowledge of available training data space.
Therefore, it is unable to request any specific training data point (input-output pair). The training data enrichment is limited to the availability of the training data space provided by the training set \( T \). Once the additional training data points are selected, the training data enrichment adds them to the current training data \( \tau_i \) and repeats the structural evolution process with the new set of training data \( \tau_{i+1} \).

### 3.8.2 Design Rational

The reasons that our proposed evolutionary learning/training process is designed in this order (hierarchical nested loops) are as follows:

1. The coefficient training must be the center of the training operation since every change in the net structure may introduce new net weights or discard some net weights. The net weights must re-learn/re-train with the current training data.
2. The training data enrichment needs the energy error record from every iteration in the coefficient training process which iterates through the structural evolution process for analysis that may lead to a conclusion about the appropriateness of the current training data. Note the scattering of the training data in the Euclidean space may not necessarily infer the true scattering in the system input-output information space. The scattering of the training data may be observed through analyzing the energy error record.

From these reasons (1 and 2), the coefficient training process (inner loop) must be prior to the structural evolution process (middle loop). The training data enrichment process (outer loop) must follow the other two processes. In this way, the net weights can adapt to the newly evolved net structure during the structural evolution. Additionally, the net weights and structure can be trained with the enriched training data to better model the input-output information during the training data enrichment.
Chapter 4

Self-Configurable Neural Net

Processor for FIR Filter Applications

A self-configurable system is one that is designed primarily for the purpose of reconfigurable control and adaptive signal processing. It evolves by restructur- and readjustments back and forth which can track the environment and the sys- tem variation in time. Processing methods and application areas include but not limited to transmission enhancement such as filtering, equalization, and noise cancellation. The performance of our proposed self-configurable neural net pro- cessor for finite impulse response (FIR) filter are compared with those of the classical FIR filters and the traditional adaptive FIR filters. The neural net processor is an autonomous system which does not need human design knowledge of the FIR filter.
4.1 Introduction and Motivation

In many applications requiring filtering, the necessary frequency response may not be known in advance. The necessary frequency response may vary with time, the system state, and the surrounding environment state. In such applications, a self-configurable neural net processor which can automatically adjust itself without human control and which can track the application system and the application environment variation in time is extremely useful.

The traditional adaptive filters can be trained to perform specific signal processing and decision making task. They can extrapolate a model of behavior to deal with new situations after having been trained on a finite and often small number of training signals or patterns [134]. However, due to the nature of traditional adaptive filter learning, iterative back propagation training can take a long time to finish [135].

In this work, we are proposing a novel self-configurable neural net processor (neural net processor) as an alternative solution to the traditional adaptive FIR filter. The neural net processor is designed primarily for the purpose of reconfigurable control and adaptive signal processing. It is more complex and difficult to analyze than non-adaptive systems and traditional adaptive systems. However, it offers the possibility of substantially increased system performance when training time limitation is enforced and input signal characteristics are unknown or time varying. Because of this, neural net processor tends to be self designing by back propagation on artificial neural network and requires no human
A significant number of back propagation methods have been proposed in the past [136, 137]. Some limited FIR filter configuration modification via genetic algorithm has been proposed [47]. However, most published approaches do not include flexible, feed-forward FIR filter structure modification. The adaptive FIR filter structure can be automatically modified with our neural net processor.

In this work, our proposed neural net processor is designed and tested for FIR filters in noise cancellation and signal equalization applications. In back propagation, the learning-rate controls the amount of gradient decent information used to update each coefficient. It directly affects how quickly the neural net processor will converge toward the unknown target process [50]. Two new complementary approaches are proposed. One, novel variable learning-rate resolves the problem of learning-rate selection which slowly increases the learning-rate from zero to the gradient with respect to the learning-rate itself. Two, novel adaptive structure accelerates the convergence speed of back propagation process by incrementally searches for a suitable structure.

The following section, Section 4.2, gives a brief overview of reconfigurable system and description of the FIR filter architecture in relation to the multi layer perceptron model. Section 4.3 presents our proposed self-configurable approach and how this approach augments the existing back propagation. Section 4.4 evaluates our neural net processor for FIR filter using the two applications. Finally, the paper concludes in Section 4.5.
4.2 FIR Filter and Neural Net

4.2.1 Finite Impulse Response Filters

In signal processing, the function of a filter is to remove unwanted parts (frequencies) of an incoming signal, such as noises, and to extract useful parts, such as components lying within a certain frequency range. There are two main kinds of filter: analog and digital; we are focusing on the digital filter which may run on a general-purpose computer, such as a personal computer, or a specialized digital signal processor chip. The prime benefit for digital signal processing over analog signal processing is flexibility [138]. In general, a digital processing system is more easily reconfigured as parameters of the problem change [139].

FIR filters are one of two primary types of digital filters used in digital signal processing applications [140]. The impulse response is finite because there is no closed loop or no feedback in the filter. The direct-form is the most commonly used and simple to implement for FIR filter with the structure of a tap delay, multiplication and addition operations, shown in Figure 4-17 and in Equation 4-5 which are called the direct-form FIR structure or a non-recursive structure. Since any FIR filter can be implemented using the direct-form, non-recursive structure, it is always possible to implement an FIR filter non-recursively [45, 141].

\[
y(n) = x(n) \cdot \left( \sum_{i=0}^{k} z^{-i} \cdot w(i) \right) + bias \cdot w(b) \tag{4-5}
\]
4.2.2 Neural Net as FIR Filter

The direct-form FIR structure can be implemented using two-layer fully connected perceptron architecture \[134, 137\] with linear activation function and an input tap delay. The direct-form structure is a feed-forward neural net, consisting of a number of neurons which are connected by weighted links. The neurons are organized in two layers, namely an input layer and an output layer, as shown in Figure 4-18. The input layer receives an external signal from the input tap delay and connected via synaptic weighted connections to all the inputs of the neurons in the output layer.

A neuron is the basic unit of the artificial neural network which has similar model in Figure 4-18. The neuron is made up of two components. The weighted sum component is responsible for calculating the summation of all products between an input, \(x_i\), and its synaptic weight, \(w_i\). The input \(x_i\) are delayed by \(z^{-1}\) operator in Figure 4-17 which is implemented by registers in neural net Figure 4-18. This summation is known as net value and is given by Equation 4-6 which
Figure 4-18: Direct-form FIR filter structure model implemented by linear two-layer perceptron

can be identified by the dotted box in Figure 4-17 and Figure 4-18. The activation function component gives the neuron’s output, evaluated based on its net value. Since our application is FIR filter and back propagation requires differentiable activation functions, the identity function is used as in Equation 4-7.

\[ net = bias \cdot w_b + \sum_{k=1}^{N} w_k \cdot x_k \] 4-6

\[ f(net) = net \] 4-7

When the input is presented to be evaluated, their input values are propagated forward to obtain the evaluated or actual outputs which are also the output of the modeled FIR filter.
4.2.3 Neural Net as Configurable, Adaptive Filter

The neural net can implement exactly the direct-form FIR filter. Adaptive filter is a filter which its coefficients can be adjusted provided a set of training inputs to closely estimate the given outputs and extrapolates to deal with new situations. However, the traditional adaptive filter is not capable of modifying its structure. The neural net processor for FIR filter is equipped with back propagation algorithm and structure modification process to support the filter coefficients adjustment and number of taps modification in a systematic way.

4.3 The Self-Configurable Approach

Back propagation on neural net requires a learning-rate which directly determines how quickly the neural net converges to a solution. Sometimes, learning-rate selection can be problematic. This work presents: a) a new variable learning-rate which resolves the issue in the learning-rate selection, and b) a new adaptive structure for neural net which accelerates the convergence speed of back propagation. Our neural net processor for FIR filter can adjust its synaptic weight coefficients efficiently and modify its neural net structure as necessary.

The approach is separated into two major processes: coefficient adjustment process and structural modification process. The coefficient adjustment process is to find the most suitable set of the synaptic weight coefficients for a given neural net structure. In addition, the structural modification process
analyzes the resulting adjusted coefficients, if needed, and modifies the neural net structure. If the structural modification process does not satisfy any terminal condition, the modified neural net will be sent for another, coefficient re-adjustment and structural re-modification. The approach keeps repeating the two processes until a terminal condition is satisfied.

Figure 4-19: Neural net processor training operation flow
4.3.1 Coefficient Adjustment Process

The coefficient adjustment processes concerns the synaptic weight coefficient values of a given neural net structure. The neural net structure model used in this work is an neural net with identity activation function described in Subsection 4.2.2. The coefficient adjustment process makes adjustments to the synaptic weight coefficient using a supervised learning method based on back propagation on neural net structure with a given input-output signal training set. It monitors the learning progress of back propagation on neural net by adjusting the synaptic weight coefficient.

Transition Phase

The coefficient adjustment process is subdivided into two phases: transition phase and stabilization phase. At the beginning of the neural net training, the learning-rate is necessary to be at a small enough value such that the gradient computation can find a direction to the back propagation's global solution more accurately. The transition phase computes the small learning-rate update, $\mu$, as in Equation 4-8

$$\mu = \alpha C + (1 - \alpha) \frac{C \cdot 1 + 1 \cdot (n - 1)}{n}$$

where $\alpha$ is a transition coefficient or a weighted value changing from 1 to 0 in each training iteration during the transition, $C = \frac{1}{N_i \cdot N_o}$ where $N_i$ and $N_o$ are the
numbers of input and output neurons respectively, and \( n \) is the number of iterations in the transition phase which determines how quickly back propagation should converge. The transition of the learning-rate’s update starts from a very small value of \( C \) to a larger value proportional to the weighted average 

\[
\lim_{n \to \infty} \frac{C \cdot 1 + 1 \cdot (n-1)}{n} \to 1.
\]

**Stabilization Phase**

Next phase is stabilization phase. Once the learning-rate update reaches \( \frac{C + (n-1)}{n} \), the neural net back propagation needs some time to stabilize itself so that the synaptic weight coefficients can converge and not overshooting the solution. The learning-rate update is kept at this value during the entire phase. The stabilization phase usually takes longer time than that of the transition phase because the synaptic weight coefficients grow or decay exponentially, so is the neural net error. From preliminary experiment, we discover the stabilization phase should take at least twice as long as the transition phase. Assuming the transition phases take \( n \) iterations and the stabilization phase takes twice as long, the total time for the synaptic weight coefficient adjustment process is \( n + 2n = 3n \) iterations.
4.3.2 Structural Modification Process

Complementary to the coefficient adjustment process, the structural modification process analyzes the neural net result from the coefficient adjustment process for tap increases or tap decreases. The FIR filter as a direct-form has a number of input tabs, Figure 4-17, in the tap delay layer. Selecting a number of tabs for an unknown FIR filter characteristic is challenging. This process uses an incremental approach of incrementing or decrementing the taps from a given structure with the aim to discover the necessary number of taps required by the unknown FIR filter characteristic. It adjusts the neural net's structure to accelerate and improve the learning process.

Tap Increment

To maintain the functionality of the direct-form FIR filter, a tap can be added to the last tap on the tap delay layer, Figure 4-17. The first tap on the tap delay layer corresponds to the zero delay operator, \( z^0 \); the second tap corresponds to the first delay operator, \( z^{-1} \), and so on. By adding another tap delay after the last tap and initialize its weights to zero will not change any functionality on the FIR filter besides delaying the output by one more time process. Assuming the given FIR filter is described within Section 4-9, the modified structure by adding another tap after the last delay is given in Section 4-10. The model in Section 4-9 and
Section 4-10 are equivalent in functionality with an extra delay in Section 4-10.

\[ y_n = x_n \cdot w_0 + x_{n-1} \cdot w_1 + \cdots + x_{n-k+1} \cdot w_{k+1} \]

\[ + x_{n-k} \cdot w_k \]  

\[ y_n = x_n \cdot w_0 + x_{n-1} \cdot w_1 + \cdots + x_{n-k+1} \cdot w_k \]

\[ + x_{n-k} \cdot w_k + x_{n-k-1} \cdot w_{k+1} \]  

where \( w_{k+1} = 0 \).

If one of the following policies is satisfied, the structural modification process will add another tap. The policy is determined by the neural net training process. Specifically, the average energy error between the neural net actual outputs and the training set desired outputs for all training iterations \((3n \text{ iterations per epoch})\) in the \(t\)-th epoch, \(E_e(t)\) is defined in Equation 4-11.

\[ E_e(t) = \frac{1}{3n} \sum_{i=1}^{3n} E_i \]

where \( E_i \) is the total error for the \(i\)-th iteration.

1. If \(E_e(t + 1) > E_e(t)\), or

2. If \(\left|\frac{E_e(t+1) - E_e(t)}{E_e(t)}\right| < \rho\), where \(\rho\) is a small number. In this work, we set \(\rho = 0.001\).

**Tap Decrement**

To decrement a tap from a given FIR filter Section 4-9 with the minimal functionality change, the last tap’s weight must be very close to zero compared to other tap
coefficients. Suppose the weight of \( x_{n-k} \) is relatively closer to zero than those of other tap coefficients; then, the \( w_k \) can be described as in Equation 4-12. The tap of \( x_{n-k} \) can be removed as it is not necessary to be there, shown in Equation 4-13, which closely approximates Section 4-9 with one less time delay.

\[
\begin{align*}
  w_k &\rightarrow 0; \quad |w_k| \ll \sum_{i=0}^{k} |w_i| \\
y_n &= x_n \cdot w_0 + x_{n-1} \cdot w_1 + \cdots + x_{n-k+1} \cdot w_{k+1}
\end{align*}
\] 4-12 4-13

If the following condition is satisfied, the structural modification process will remove the last tap. The condition is determined by the last tap weight update during the neural net training, shown in Equation 4-14

\[
\left| w_k \right| < \frac{1}{N_i \cdot N_o} \sum_{i=1}^{k} |w_i| \quad \text{and} \quad \left| \frac{w_k(t+1) - w_k(t)}{w_k(t)} \right| < \rho
\] 4-14

where \( \rho \) is a small number. In this work, we set \( \rho = 0.001 \).

### 4.3.3 Terminal Condition

The structural modification process decides on one of three choices: increment, decrement, or no-change. It checks for the tap incremental condition first, then
the tap decrement condition. If the decision is either the tap increment or de-crement exclusively, the structural modification process modifies the neural net struc-
ture. If neither the tap incremental nor the tap decrement condition is satis-
fied or both conditions are satisfied, the structural modification process does not
change the structure (no-change decision). Since the tap increment and decre-
ment conditions are independent, they can be satisfied at the same time. Af-
ter the decision action, the structural modification process checks the terminal
condition. If the terminal condition is not satisfied, the structure modification
process sends the modified neural net processor to repeat the two processes for
another restructuring.

One terminal condition is two consecutive rest decisions which mean that
the coefficient and the structure of the neural net processor have converged enough.
Another terminal condition for the neural net training is the following three con-
secutive structure modification decisions: decrement, increment, and decrement,
respectively. The three consecutive decisions indicate the structure of the neu-
ral net processor oscillation sequence. The oscillation implies the two neural
net configurations are equivalent in term of functionality regardless of time de-
lay. Between two equivalent neural net processors, the one with least number of
neurons is a more efficient neural net processor. Thus, the least neuron number
neural net processor is the unknown FIR filter estimation.
4.4 Experimentation Results

We are going to demonstrate our neural net processor for FIR filter with two applications: noise cancellation and signal equalization. Both of these applications lie within the system discovery problem class which can be solved with the traditional adaptive filter with some limitations or using our proposed neural net processor which is more flexible. Other potential applications can be found in [139]. The goal of neural net processor for FIR filter is to identify, estimate, and extrapolate a given unknown FIR filter characteristic from a given input-output signal training set. This is done with our proposed variable learning-rate approach and the proposed adaptive structural modification. Hence, the response of the neural net processor for FIR filter is improved as well.

4.4.1 Noise Cancellation

The following is an example of the noise cancellation application setup. Suppose we have a low frequency control input signal transmitting from a control center at distance in the analog domain. The signal travels through a cable possibly intervening with power supply lines or other cables. The power supply current acts as a noise signal source. Assuming the power-supply has the frequency of 60 Hz, Figure 4-22(a). The noise source is able to interfere with our control signal through the cable shields which act as an unknown signal processor. Figure 4-20(left side), the processed noise interference with our control signal result into a contaminated signal, Figure 4-22(b). At our target receiver, Figure 4-20(right
side), the contaminated signal and the noise source are digitized with an analog-to-digital converter (ADC). The digital noise source is filtered with a well-trained neural net processor producing the estimate of the unknown processed noise. The estimate signal is subtracted out from the digital contaminated signal as the 180° phase difference. The output digital signal closely estimates our original control signal.

![Noise cancellation model](image)

Figure 4-20: Noise cancellation model

The training process of the neural net processor described by the block diagram, Figure 4-21, is that the neural net processor extracts an estimate of the simulated unknown noise signal processor. Given a simulated sample input noise signal and a simulated unknown filtering process, the neural net processor can learn about the unknown filter operation such that it can closely estimate the unknown direct-form filter's structure. The objective is to change or adapt the synaptic weight coefficients and the neural net structure of the neural
net processor to match as closely as possible the response of the simulated unknown filtering process. The simulated unknown filtering process and the neural net process the same simulated given input signal producing the expected (or desired) output signal and the actual output signal. The error between the two outputs is measured and fed back to the neural net trainer for coefficient and structural adjustment. After repeatedly adjusting each weight and structure, the neural net processor should converge; that is, the difference between the desired output signal and the actual output signal should get smaller and smaller.

![Diagram of neural net processor training model for noise cancellation application](image)

Figure 4-21: Neural net processor training model for noise cancellation application

We are testing four different noise signal shapes: saw, square, triangle, and sine noise waveforms, with all signals being at 60 Hz frequency. Table 4-3 shows the result from our experimentation of noise cancellation by injecting the different noises to a low frequency signal (1 Hz). The saw noise waveform takes the longest iteration time in the neural net training at 540 iterations or 18 epochs (30 iterations is 1 epoch). The learning process recovering the unknown FIR filter requires 16 tabs. All neural net training sessions are preset with 10 iterations for the
transition period \((3 \times 10 = 30\) iterations per epoch\) and the arbitrary starting FIR structure is at 3 input taps delay.

We have also run a comparison experiment for traditional adaptive FIR filter. The comparison experiment uses the same initial setting and the same number of training iterations as the neural net processor experiment. The traditional adaptive FIR filter has no ability to change the numbers of taps on the FIR
Table 4-3: Result of noise cancellation neural net processor learning

filter. Thus, the number of FIR taps is always 3 which is the ideal traditional FIR filter configuration. The result is shown in Table 4-4. The Saw signal and Square signal training produces very close final energy error and final learning-rate with the previous experimentation showing the neural net processor can achieve the same performance as the ideal traditional FIR filter configuration. However, the Triangle and Sine signals for the traditional adaptive FIR filter have much higher final energy error. This means, given the same amount of training time, the neural net processor can better approximate the unknown FIR filter than the traditional adaptive FIR filter.

Table 4-4: Noise cancellation on ideal traditional adaptive FIR filter configuration
4.4.2 Signal Equalization

Another application of the neural net processor is data transformation and its inverse transformation, specifically signal equalization in digital signal processing. The following is an example of the signal equalization setup connecting the equalizer in series with a transmission line to alter the frequency response characteristics of the line.

With the same experimental setup as in Noise Cancellation application, given a simulated sample input data and a simulated unknown attenuation, the neural net processor can learn the inverse of the simulated attenuation operation such that it can boost or cut the distortion signal to closely estimate the original signal. The idea of the neural net training behind the block diagram, Figure 4-23, is that the neural net processor extracts an estimate of the desire original input which can be used as the original data.

![Diagram of Neural net processor training model for signal equalization application](image)

Figure 4-23: Neural net processor training model for signal equalization application

Suppose we are transmitting data signal through a channel from distance.
The communication channel can possibly be attenuated through the environment. The data signal at the receiving end might be distorted. The distorted data signal is digitized by an ADC and fed to the neural net processor. The neural net processor processes the digital distorted data signal producing the estimate of the original data signal. We are testing four different data signal shapes: saw, square, triangle, and sine waveforms at 1 Hz. Table 4-5 shows the result from our experimentation of signal equalization by recovering the estimate of the original data signal from the receiving distorted signal. All signal equalization training sessions are preset similarly to that of the Noise Cancellation application.

<table>
<thead>
<tr>
<th>Original Signals</th>
<th>Iterations</th>
<th>Taps</th>
<th>Energy Error</th>
<th>Learning-Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saw Wave</td>
<td>120</td>
<td>2</td>
<td>0.641</td>
<td>0.1101</td>
</tr>
<tr>
<td>Square Wave</td>
<td>120</td>
<td>2</td>
<td>0.761</td>
<td>0.0733</td>
</tr>
<tr>
<td>Triangle Wave</td>
<td>60</td>
<td>3</td>
<td>1.758</td>
<td>0.0598</td>
</tr>
<tr>
<td>Sine Wave</td>
<td>60</td>
<td>3</td>
<td>1.970</td>
<td>0.0167</td>
</tr>
</tbody>
</table>

Table 4-5: Result of signal equalization neural net processor learning

Table 4-6 shows the result from the traditional adaptive FIR filter on the same experimental set up. The number of FIR taps is fixed at 3 which is the ideal traditional FIR filter configuration. Though the final energy error of the traditional adaptive FIR filter is smaller, the final learning-rate of neural net processor is slightly higher. If we give the training more training time, the neural net processor can produce a closer estimate of the unknown attenuation process due to the larger learning-rate.
<table>
<thead>
<tr>
<th>Original Signals</th>
<th>Iterations</th>
<th>Energy Error</th>
<th>Learning-Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saw Wave</td>
<td>120</td>
<td>0.469</td>
<td>0.1081</td>
</tr>
<tr>
<td>Square Wave</td>
<td>120</td>
<td>0.570</td>
<td>0.0715</td>
</tr>
<tr>
<td>Triangle Wave</td>
<td>60</td>
<td>1.759</td>
<td>0.0598</td>
</tr>
<tr>
<td>Sine Wave</td>
<td>60</td>
<td>1.971</td>
<td>0.0167</td>
</tr>
</tbody>
</table>

Table 4-6: Signal equalization on ideal traditional adaptive FIR filter configuration

4.4.3 Comments on the Results

The current state of our work is on the simulation of the neural net processor for FIR filter. The neural net processor implementation on an FPGA will be in the future work. Therefore, the results presented in this work do not contain any FPGA overheads or reports on area, speed, nor power consumption.

The neural net processor for FIR filter can be applied to any signal frequency, not limited to 1 Hz or 60 Hz frequencies as presented in Subsection 4.4.1 and Subsection 4.4.2. The approach is independent of the signal frequency since the operation speed is determined by the hardware system components. However, with the neural net processor approach, a training iteration may take longer to complete compared to the traditional adaptive filter. Therefore, although both techniques use the same number of iterations, the traditional one might be faster. Nevertheless, this is the trade off for the neural net processor autonomous design ability.
4.5 Discussion and Analysis

This work presents a self-configurable neural net processor for finite impulse response filter to solve the problem class of system identification discovery which can be used in many applications. Our proposed neural net processor (adaptive learning-rate and structure modification) have demonstrated improvements in the implementation of the classical FIR filters and the traditional adaptive FIR filters in noise cancellation and signal equalization. We have shown that an acceptable variable learning-rate is achievable with the proposed learning-rate update and the structure modification at a reasonable cost. Using incremental structure modification, it is possible to find a global optimal FIR direct-form structure which the characteristic of the neural net processor closely estimates an unknown FIR filter. Finally, the neural net processor for FIR filter is totally autonomous which does not require human in designing an FIR filter.
Chapter 5

Configurable FIR Filter Scheme
based on an Adaptive Multilayer
Network Structure

In this work, we present a design technique of configurable FIR filter architecture based on neural net like (multilayer network) structure. This architecture is a generalization of the configurable adaptive FIR filters and can be implemented on an FPGA. The design is based on a combination of pipelining and folding schemes at the multiplication-addition component level, network cell level, and network layer level. The proposed configurable multilayer network technique can reduce latency and hardware requirements while increasing the throughput of the filter. The weighted input connections, the network cells, and the number of network layers are configurable to fit filter design requirements. The configurable pipelining/folding scheme and the parameter setting characterize
the proposed FIR filter architecture, FPGA space, and operation timing requirements. The configurable architecture is compared with several traditional FIR filter structures regarding hardware and time complexity. The potential applications of the proposed architecture are also discussed with respect to traditional adaptive FIR filter performance.

5.1 Introduction and Motivation

Nowadays, digital signal processing (DSP) is used in a wide variety of real-time application [142, 135, 143] and is playing an important role in the digital revolution. Finite impulse response (FIR) digital filters are the most fundamental DSP components. FIR filters [135] have the advantage of stability and easy implementation, but the large number of filter taps leads to excessive hardware complexity. Pipelining techniques [144, 145, 146] can increase throughput of FIR filters. On the other hand, several FIR filter folding techniques [147, 47, 145] have been proposed as a means of reducing hardware when the processing throughput required by the application is less than the throughput at which the design can operate.

A bit-level folding design [144, 147] proves to be more hardware efficient but is not configurable. Reconfigurable FIR filter architectures have been presented in [148, 47, 149, 146, 150] but do not support adaptive filter behavior where the filter coefficients can be trained to adjust its response to a specific filter specification.
A number of algorithms for digital filter design have been proposed [135, 143]. The methods are based on Fourier series, Remez exchange method, frequency sampling, and equiripple designs. All these methods can only be used to design standard filters which are the product of a time discrete filter and the transfer function of the noise shaper. The methods generate a large number of coefficients and over-specifications.

In this work, we propose a multilayer network structure (MLNS) FIR filter architecture as an alternative solution to the existing techniques. The MLNS architecture can implement FIR filters by configurable pipelining-folding and/or parallel-serial schemes. The MLNS filter is primarily designed for configurable adaptive FIR filters which can be realized on an FPGA—reconfigurable number of filter taps, filter tap coefficients, and filter architectures. This hierarchical structure leads to higher throughput, lower latency, and reasonable hardware complexity. Moreover, the MLNS can potentially support both linear and non-linear FIR filter operations.

The organization of the paper is as follows. In Section 5.2, we provide a comprehensive introduction of traditional FIR filters, filter implementation techniques, and how the MLNS architecture is related to FIR filters. Implementation techniques for the proposed MLNS architecture are presented in Section 5.4. Also, in Section 5.5, a comparison among various MLNS filter implementations with a straight pipelined filter and a traditional folded filter are evaluated. Additionally, the proposed MLNS architecture is demonstrated by implementing in
FPGA MLNS adaptive FIR filters which are compared with the traditional adaptive FIR filter. Finally, we discuss some potential applications of the MLNS architecture in Section 5.6; and, the paper concludes in Section 5.7.

5.2 Background and Related Works

5.2.1 Traditional FIR Filters

Finite impulse response (FIR) filters are one of two primary types of digital filters used in digital signal processing (DSP) applications [135, 143]. The impulse response is finite because there is no closed loop or no feedback in the filter. The direct-form, a non-recursive structure or no feedback loop, is the most commonly used and simple to implement for FIR filter with input tap delays, multiplications and additions, as shown in Figure 5-24 and in Equation 5-15. Since any FIR filter can be implemented using the direct-form, it is always possible to implement an FIR filter non-recursively [135].

\[
y_n = x_n \cdot \left( \sum_{i=0}^{k-1} z^{-i} \cdot w_i \right) + bias \cdot w_b
\]

5-15

FIR filters can be realized with various architectures incorporating the operations of the filter taps multiplications and their additions. With some internal registers, the computation unit that performs this operation is a multiplier combined with an adder, called a multiplier-adder component (MAC), shown in
Figure 5-24: Direct-form FIR filter structure equivalent to transfer function in Equation 5-15

A dotted box (1 stage) of Figure 5-24. The MAC controller carries out the preset number of FIR tap operations with the input $x_n$ read from the tap delays, the tap coefficients (weights) already stored in the memory, and the output $y_n$ written back in the memory. In one implementation approach, MAC units can implement FIR filters by straightforward cascading with pipelining technique. Another implementation approach is to fold the cascaded MAC units into a single MAC with a time-multiplexed folding controller.

Assuming the multiplication and addition operation time in the MAC are the same and they perform their operations simultaneously saving their results in the internal registers, the time for the MAC operation will be referred to as one MAC cycle. The traditional folding technique requires only one MAC unit to produce one final result at the output within $k$ MAC cycles latency, where $k$ being the number of filter taps shown in Figure 5-24. On the other hand, the straight pipelined filter needs $k$ multipliers and $k$ adders (or $k$ MAC units). With pipelining, a final result is produced in one MAC cycle; however, the latency remains at
5.2.2 Transposed and Hybrid FIR Filter Architectures (Related Works)

Various research works [144, 147, 47, 145, 146] have been proposed on how best to implement \( k \)-tap FIR filter, shown similar to Figure 5-24. One architecture describes FIR filter with the transposed-form [135]. Realizing Equation 5-15 as a sum of successively delayed inputs multiplied by the appropriate filter coefficient yields the direct-form implementation as shown in Figure 5-24. Transposing the direct-form implementation yields the transposed-direct (or simply the transposed-) form implementation. The transposed-form FIR filters are canonical in that it uses \( k \) coefficient multipliers, \( k \) additions, and \( k \) multiplications (or \( k \) MAC units) for \( k \)-tap FIR filters. It also has the latency of \( k \) MAC cycles if implemented using either folding technique or pipelining.

Another FIR filter architecture is in hybrid-form [151, 135] when combining the transposed-form with the direct-form to implement the same canonical filter. Intuitively, the hybrid-form is obtained from the direct-form by moving a minimum number of registers from the input path to the summation path to satisfy the cycle-time requirement. The hybrid-form uses the same number of MAC units as the direct- and transposed- forms. The theoretical latency timing of hybrid-form is similar to the previous two architectures as \( k \) MAC cycles for folding technique and pipelining.
There are other forms of FIR filters which are not mentioned in this work. Nonetheless, they are implementation variations to obtain the minimum time and hardware requirements for specific applications [147, 152]. The other variations of FIR filter forms can be partially folded, hierarchical, or parallel [153]. This work focuses on digital filter operation; analog filters or digitization effect, such as distributed arithmetic technique [154, 155] which is analogous to the transpose and hybrid forms, are not considered.

5.2.3 Multilayer Network Structure as FIR Filters (Contributions)

Alternative to the existing FIR filter architectures, we propose the multilayer network structure (MLNS) to implement FIR filters with partially connected network topology and input tap delays. Our proposed structure is similar to a multilayer neural net structure [16] with a linear or identity activation function. The MLNS architecture consists of multiple network layers (NLs); and, each network layer consists of multiple network cells (NCs), as shown full scale in Figure 5-25. The first NL (input layer) receives external inputs from the input tap delays. Assuming each NL has \( p \) NCs with \( k \) weighted input connections, the length of the input tap delays is \( K = p \cdot k \). The outputs of the input layer are connected via weighted connections to a set of network cells in the next network layer and propagate through all layers in the network. The final results are captured at the outputs of the last NL (output layer).
A network cell (NC) is the basic unit of the MLNS architecture whose structure is shown in Figure 5-26. The NC is made up of two components: the weighted component and the summation function component (similar to a neuron in a neural net with an identity activation function [16]). The weighted component is responsible for calculating the product of all the inputs and their corresponding weights, which is equivalent to the multiplication triangle boxes in Figure 5-24. The summation function computes the summation of all the weighted products which is equivalent to the series of addition circles in Figure 5-24.

Since the structure of the network cells coincide with that of the FIR filter shown in Subsection 5.2.1 and Subsection 5.2.2, the network cells can be implemented with the same approaches for the FIR filters. In this work, we are focusing on the implementation with pipelining and folding techniques.
5.3 Formulation of MLNS as FIR Filters

The MLNS computation as FIR filters can be formulated into mathematical equations for more accuracy. Assume, as shown in Figure 5-25, the MLNS has $m$ NLs and each NL has $p$ NCs with each NC having $k$ weighted input connections. The $k$ weights are indexed as $w_{ij}^h$, indicating the weight from the $i$-th NC of the $(h - 1)$-th NL to the $j$-th NC of the $h$-th NL. Since the outputs of the $h$-th NL become the inputs of the $(h + 1)$-th NL next layer, the inputs of the $(h + 1)$-th NL or the outputs of the $h$-th NL are also indexed as $a_{ij}^h$, indicating the output of the $j$-th NC in the $h$-th NL to the $i$-th NC of the $(h + 1)$-th NL, which can be recursively calculated from the previous NL's outputs.

In this work, we assume the full and balance tree structure for the MLNS architecture which means the number of NCs in all NLs are equal to $p = k$. For a particular output computation, one output of the first NC in the first NL can be computed as in Section 5-16, which is similar to Equation 5-15 with different variable notations. The $a_{ij}^0$ indexes the outputs of the $0$-th NL which is the FIR
filter inputs stored in the input tab delays of Figure 5-24 and Figure 5-25. The output shown in Section 5-16 is equivalent to a result of $k$-tap FIR filters.

$$a_{11}^1 = \sum_{s=1}^{k} w_{1s}^1 \cdot a_{1s}^0$$

$$= w_{11}^1 \cdot a_{11}^0 + w_{12}^1 \cdot a_{12}^0 + \cdots + w_{1k}^1 \cdot a_{1k}^0$$  \hspace{1cm} 5-16

The other NCs’ outputs in the first NL are computed in the same fashion with different weights and inputs indexing. The outputs of the next NL are computed in similar manner since the MLNS is hierarchical. The outputs of the previous NL become the inputs for the next NL. An output computation of the second NL can be written out as the fully expanded formulation in Section 5-17.

$$a_{21}^2 = \sum_{t=1}^{k} \sum_{s=1}^{k} w_{1t}^2 \cdot w_{ts}^1 \cdot a_{1t}^1$$

$$= \sum_{t=1}^{k} w_{1t}^2 \cdot \left( \sum_{s=1}^{k} w_{ts}^1 \cdot a_{ts}^0 \right)$$

$$= w_{11}^2 \cdot \left( w_{11}^1 \cdot a_{11}^0 + w_{12}^1 \cdot a_{12}^0 + \cdots + w_{1k}^1 \cdot a_{1k}^0 \right)$$

$$+ \cdots$$

$$+ w_{1k}^2 \cdot \left( w_{k1}^1 \cdot a_{k1}^0 + w_{k2}^1 \cdot a_{k2}^0 + \cdots + w_{kk}^1 \cdot a_{kk}^0 \right)$$  \hspace{1cm} 5-17

Section 5-17 can be further simplified by distributing the weights of the second NL, $w_{1t}^2$, into the output formulations of the first NL computation. The
simplified formulation of Section 5-17 is shown in Section 5-18 with a new arrangement. All the inputs of Section 5-18 are FIR filter inputs from the input tap delays.

\[
a_{11}^2 (w_{11}^2 \cdot w_{11}^1) \cdot a_{11}^0 + \cdots + (w_{1k}^2 \cdot w_{1k}^1) \cdot a_{1k}^0 \\
+ \cdots \\
+ (w_{k1k}^2 \cdot w_{k1k}^1) \cdot a_{k1k}^0 + \cdots + (w_{kkk}^2 \cdot w_{kkk}^1) \cdot a_{kkk}^0
\]

5-18

The multiplications of the weights are considered as another weight or another FIR tap coefficient. Thus, the output of Section 5-18 is essentially one result of \(k^2\)-tap FIR filters since \(k \cdot k = k^2\) filter tap inputs are presented. The tap coefficients of the \(k^2\)-tap FIR filter are calculated as the products of network weights. Section 5-18 can be rewritten in the FIR filter form as in Equation 5-19.

\[
a_{11}^2 = c_{11} \cdot a_{11}^0 + c_{12} \cdot a_{12}^0 + \cdots + c_{kk} \cdot a_{kk}^0
\]

5-19

where the tap coefficients are \(c_{11} = w_{11}^2 \cdot w_{11}^1, c_{12} = w_{11}^2 \cdot w_{12}^1, \ldots\).

The computation of the next NL is similar to the previous layers. The output equations can be simplified and rearranged such that all inputs of the equations are the FIR filter tap inputs from the input tap delays. At the third NL, one output is equivalent to one result of \(k^3\)-tap FIR filters and so on.
5.4 The MLNS Architecture

The multilayer network structure (MLNS), shown in Figure 5-25, evaluates the presented inputs at the input tap delays by feed forward propagating the input values through the input layer, all the hidden layers, and finally to the output layer as briefly described in Subsection 5.2.3. The evaluation within a network layer (NL) consists of a set of multiple network cells (NCs); and, an NC consists of multiple weighted input connections (MAC). The MLNS can be implemented using either component pipelining or folding techniques or a combination of the two. Since the MLNS consists of three component levels—network layer, network cell, and weighted input connections (MAC)—one level can be folded while the others are pipelined or vise versa. Additionally, at each level, the computation can be carried out with a single unit serially or with multiple units in parallel.

5.4.1 Implementation at MAC Level

The NC consists of $k$ associated number of weighted input connections which is similar to $k$-tap FIR filters having associated number of taps. The NC computation coincides with the computation of FIR filters as described in Subsection 5.2.3. Thus, the implementation of FIR filter can be applied to the NC. The pipelining technique requires $k$ MAC units to produce a result within one MAC cycle latency. On the other hand, the folding technique requires one MAC unit to produce a final result in $k$ MAC cycles latency.
5.4.2 Implementation at Neural Cell Level

The NL architecture can be implemented with multiple NCs in parallel or with a single NC serially. The implementation of the NL module with multiple NCs in parallel is shown in Figure 5-25. All parallel NCs are operating simultaneously to compute the partial FIR filter operation. Therefore, the FPGA space required for the NL in parallel mode is the number of NCs in parallel, $p$. The NL operation time, however, is only the operation latency time of one NC.

To show the NL operation in serial scheme, the operation flow of the serial NC architecture is illustrated in Figure 5-27 with a single NC in the dashed box. An NL embedded controller does context switching for the NC to carry out one NC operation for different weight and input sets. Thus, the NL requirement in serial mode is the latency time for one NC to perform $p$ NC operations or $p$ NC cycles.

![Figure 5-27: Serial neural cell (NC) architecture](image)
5.4.3 Implementation at Neural Layer Level

The MLNS architecture can also be folded or pipelined at the NL level. The pipelined architecture at NL level is shown in Figure 5-25. Multiple copies of NLs pipeline the inputs from the tap delays passing through each network layer and produce a final result at the last NL. For the architecture in Figure 5-25, the pipelining technique requires \( m \) NLs with the operation latency time of \( m \) NLs cycle.

On the other hand, the MLNS architecture essentially consists of a repetition of the NLs which can be implemented by another time-multiplexed operation. The MLNS architecture can be folded using one network layer with a time-multiplex controller as shown in Figure 5-28. The NL component can be implemented as described in Subsection 5.4.2. The folded architecture requires only one NL to produce a final output in \( m \) NL cycles latency. The total FPGA space and operation time requirement depends on the implementation of the NL component.

5.5 Complexity Analysis

FIR filter implementations have been presented in various works in the literature, some of which are mentioned in Section 5.2. We consider a straight pipelined filter, a traditional folded filter, and various combinations of configurable MLNS FIR filters presented in Section 5.4.
5.5.1 MAC Usage, Latency, and Design Throughput Analysis

Assume we are implementing $K$-tap FIR filters, similar to Figure 5-24; and, the multiplier and the adder of the MAC unit requires the same amount of clock cycles to complete their operations which is referred to as one MAC cycle. For the proposed MLNS architecture in Table 5-7 and from this point on, we are assuming that the MLNS architecture is a minimum configuration for a given filter design. A minimum configuration refers to an MLNS filter design that has the exact number of required taps or the least amount of extra unused taps.

The MAC usage, latency, and design throughput data is presented in Table 5-7 including the number of MAC usage column, the latency column, and the design throughput column. The spaces required by these architectures are measured in the number of MAC units used in implementing the architecture.
The latency measures the delay number of MAC cycles from when an input is initiated to when the corresponding result is produced. We measure the design throughput by the number of filter results (excluding all intermediate results from pipelining and folding schemes) produced within $T$ MAC cycles.

For Table 5-7 and the upcoming analysis, we associate labels and filter implementations for the ease of references.

- “A” refers to the straight pipelined filter implementation
- “B” refers to the traditional folded filter implementation
- “C” refers to the MLNS filter implementation with folded MACs, parallel NCs, and pipelined NLs
- “D” refers to the MLNS filter implementation with folded MACs, parallel NCs, and folded NLs
- “E” refers to the MLNS filter implementation with folded MACs, serial NCs, and pipelined NLs
- “F” refers to the MLNS filter implementation with folded MACs, serial NCs, and folded NLs
- “G” refers to the MLNS filter implementation with pipelined MACs, parallel NCs, and pipelined NLs
- “H” refers to the MLNS filter implementation with pipelined MACs, parallel NCs, and folded NLs
• “I” refers to the MLNS filter implementation with pipelined MACs, serial NCs, and pipelined NLs

• “J” refers to the MLNS filter implementation with pipelined MACs, serial NCs, and folded NLs

<table>
<thead>
<tr>
<th>MAC Units</th>
<th>Latency</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$K$</td>
<td>$T - K + 1$</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>$K$</td>
</tr>
<tr>
<td>C</td>
<td>$m \cdot p &lt; K$</td>
<td>$m \cdot k &lt; K$</td>
</tr>
<tr>
<td>D</td>
<td>$p &lt; K$</td>
<td>$m \cdot k &lt; K$</td>
</tr>
<tr>
<td>E</td>
<td>$m &lt; K$</td>
<td>$m \cdot p \cdot k \geq K$</td>
</tr>
<tr>
<td>F</td>
<td>$1 &lt; K$</td>
<td>$m \cdot p \cdot k \geq K$</td>
</tr>
<tr>
<td>G</td>
<td>$m \cdot p \cdot k \geq K$</td>
<td>$m \cdot k &lt; K$</td>
</tr>
<tr>
<td>H</td>
<td>$p \cdot k = K$</td>
<td>$m \cdot k &lt; K$</td>
</tr>
<tr>
<td>I</td>
<td>$m \cdot k &lt; K$</td>
<td>$m \cdot p \cdot k \geq K$</td>
</tr>
<tr>
<td>J</td>
<td>$k &lt; K$</td>
<td>$m \cdot p \cdot k \geq K$</td>
</tr>
</tbody>
</table>

Table 5-7: MAC usage, latency, and design throughput analysis comparison ($K = p \cdot k$ and $m < (p = k)$)

The overhead of all pipelining schemes is only a pipeline controller for internal cascaded components. The folding technique also contains a time-multiplexed
controller of the internal folded components. The overhead operation time and FPGA space complexity depends on the design of the controllers. With a large enough MLNS design, the controller complexity is much less complex than the complexity of the whole implementation. Often, FIR filters are designed with large tap number which results in huge implementation. Therefore, we assume the MLNS architecture is for large FIR filters and ignore the controller overhead complexities.

5.5.2 Sample Synthesis Result

Selected six filter designs from Table 5-7 are implemented in Verilog Hardware Definition Language (Verilog-HDL). They are compiled, synthesized, simulated using Quartus II 5.1 Service Pack 2 Web Edition Full from Altera Corporation. The compilation is targeted for generic Stratix devices. The hardware usage or area refers to the number of logic elements which are required by the Quartus software compiler and synthesizer to implement the filter designs.

The synthesis results from the software compilation-synthesis reports are shown in Figure 5-29. The figure contains the area required to realize the filter designs on the targeted FPGA platform versus the number of filter taps, $K$, for the six FIR architectures.

For this experimental result, all synthesized filter designs are simulated on their net-list with an external clock of 4 ns period or 250 MHz frequency. The
Figure 5-29: Hardware complexity comparison for the six filter implementations.

Throughput is defined as the maximum number of results (excluding intermediate results in pipelining and folding schemes) a filter design can achieve within one second. The simulation results of the throughput for six filter designs are shown in Figure 5-30.

Figure 5-30: Throughput comparison for the six filter implementations.
From the test synthesis and the simulation results targeted the specified Stratix FPGA device, the straight pipelined filter performs the filtering operation at the highest throughput while it suffers the most from hardware usage. On the contrary, the traditional folded filter requires the minimum hardware area, but it carries out the filtering operation at the lowest throughput. The proposed MLNS filter implementation is the compromise of the previous two approaches. It requires less hardware than the straight pipelined filter, and it operates at higher throughput than the traditional folded filter even with the extra controller overhead taken into account.

Nonetheless, the synthesis results are subjected to a particular FPGA software and device. If we are to change the synthesis software options, such as synthesis optimization parameters, we might obtain slightly different results. Moreover, FPGA platforms are optimized quite differently among commercial companies. The performance of the actual architecture on different FPGA devices is subject to a particular FPGA platform.

### 5.5.3 Sample Simulation Result

Our proposed MLNS architecture performance is demonstrated with adaptive filters which are regularly used in signal cancellation, channel equalization, and noise reduction. All these applications lie within the system discovery problem class which can be solved with the traditional adaptive filter with some limitations or using our proposed MLNS architecture for adaptive FIR filter which is
more flexible. Other potential applications can be found in [139, 156]. The goal of MLNS adaptive FIR filter is to identify, estimate, and extrapolate a given unknown FIR filter characteristic from a given input-output signal training set. This is done with variable learning-rate approach and adaptive structural modification [8]. Hence, the response of the MLNS adaptive FIR filter is improved as well.

The training process of the MLNS filter described by the block diagram, Figure 5-31, is that the MLNS filter extracts an estimate of the simulated unknown filtering process. Given a simulated input signal and a simulated unknown filtering process, the MLNS filter can learn about the unknown filter operation such that it can closely estimate the unknown filter's structure and coefficients. The objective is to change or adapt the synaptic weight coefficients and the neural net-like structure of the MLNS filter to match as closely as possible the response of the simulated unknown filtering process. The simulated unknown filtering process and the MLNS filter evaluate the same simulated input signal producing the expected (or desired) output signal and the actual output signal. The error between the two outputs is measured and fed back to the MLNS trainer for coefficient and structural adjustment. After repeatedly adjusting each weight and structure, the MLNS should converge; that is, the difference between the desired output signal and the actual output signal should get smaller and smaller.

We run a baseline experiment for traditional adaptive FIR filter with variable learning rate (no structural modification). The baseline experiments have
the same initial filter setting (8 initial taps) and the same number of training iterations (250 iterations) for the traditional adaptive FIR filter. The traditional adaptive FIR filter has no ability to change the numbers of taps; thus, the number of FIR taps is always 8 taps. The result is shown in Table 5-8 for low pass (LP), high pass (HP), band pass (BP), and band stop (BS) filters respectively. We do not provide comparisons with previous works because the existing published results on reconfigurable FIR filters are very limited.

<table>
<thead>
<tr>
<th>Noise</th>
<th>Iterations</th>
<th>Taps</th>
<th>Relative Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ideal</td>
<td>-</td>
<td>13</td>
<td>0.000</td>
</tr>
<tr>
<td>MLNS (LP)</td>
<td>250</td>
<td>12</td>
<td>1.383</td>
</tr>
<tr>
<td>Traditional (LP)</td>
<td>250</td>
<td>8</td>
<td>2.790</td>
</tr>
<tr>
<td>MLNS (HP)</td>
<td>250</td>
<td>13</td>
<td>1.719</td>
</tr>
<tr>
<td>Traditional (HP)</td>
<td>250</td>
<td>8</td>
<td>5.027</td>
</tr>
<tr>
<td>MLNS (BP)</td>
<td>250</td>
<td>12</td>
<td>1.246</td>
</tr>
<tr>
<td>Traditional (BP)</td>
<td>250</td>
<td>8</td>
<td>2.421</td>
</tr>
<tr>
<td>MLNS (BS)</td>
<td>250</td>
<td>13</td>
<td>0.946</td>
</tr>
<tr>
<td>Traditional (BS)</td>
<td>250</td>
<td>8</td>
<td>1.557</td>
</tr>
</tbody>
</table>

Table 5-8: Result of low pass (LP), high pass (LP), band pass (BP), and band stop (BS) adaptive filters
All of the MLNS adaptive FIR filter trainings produce better relative error than the traditional adaptive FIR filters. This indicates the MLNS filter is a closer estimator than the traditional ones. For the high pass and band stop MLNS adaptive FIR filters in the sample case, the final tap number after 250 training iterations are 13 which is the same as the ideal FIR filter that we are targeting. However, for low pass and band pass filters, the MLNS filters produce FIR filters with 12 taps which are smaller than the targeted ideal filters with 13 taps.

5.6 Potential Applications

The proposed MLNS architecture for FIR filters can be configured to implement various architectures from fully pipelined/parallel to fully folded/serial and the combinations by the configuration of pipelining or folding scheme at the MAC and NL levels and parallel or serial scheme at the NC level. In addition, the number of NLs, the number of NCs, and the number of NC’s weighed input connections are configurable to fit targeted filter hardware and timing requirements. The traditional folded FIR filters can be configured from selecting folded NL, serial NC, and folded MAC implementation. The straight pipelined FIR filter can also be configured. Moreover, other combinations of pipelining and folding scheme are made possible, such as partially folded and hybrid FIR filters.

The proposed MLNS filter can be implemented on an FPGA where adaptability can be implemented by the re-configurability of this architecture. Examples of adaptive MLNS filters are low pass, high pass, band pass, and band stop
filters. Neural net-like structures can implement adaptive filter which contains coefficients that are updatable by adaptive algorithms to improve or optimize the filter's response to a desired performance criterion. The MLNS is considered as the filter which applies the require processing on the incoming signal which is to be filtered; and, a back propagation which adjusts the coefficients of the filter improves the MLNS filter performance or can change to track filter environment.

The key difference of our proposed MLNS adaptive FIR filter and the traditional adaptive FIR filters is the MLNS filter has the ability of perform adaptive structural modification. Unlike the traditional adaptive FIR filter, the MLNS filter can analyze the given filter training process whether the current structure can produce appropriate filter response. The MLNS can structurally modify itself by adding an extra tap or removing an unnecessary tap as well as adjusting the weight coefficients.

Moreover, the proposed MLNS filter can be further implemented for non-linear filters by configuring the non-linear activation functions of the network cells. A non-linear filter is a signal-processing device whose output is not a linear function of its input. We will be investigating into the limitations of our proposed MLNS for non-linear filter applications. Examples of non-linear filters include phase-locked loops, detectors, and mixers. One practical reason to use nonlinear filters instead of linear ones is that linear filters may be too sensitive to a small fraction of large observations at the input. In practice, linear filters are only useful as pre-processing and post-processing filters. They lack the capabilities for
performing image analysis.

We present the synthesis results of our MLNS FIR filter architecture up to 124 taps on FPGA in Figure 5-29 and Figure 5-30. The sample simulation results in Subsection 5.5.3 illustrates our MLNS FIR filter application up to 13 taps; certainly, we can extend the simulation up to what we can synthesize on FPGA. The real limitation for practical realizations is the FPGA hardware size (number of logic elements) which can support our higher order MLNS FIR filters.

## 5.7 Discussion and Analysis

In this work, a configurable MLNS FIR filter architecture is proposed. The MLNS architectures can be pipelined at the network layer and MAC levels to produce high throughput and low latency. On the other hand, its multi-folded hierarchical structure requires less FPGA area. The MLNS architecture is configurable and can be implemented in a combination of pipeline-folded and parallel-serial schemes on an FPGA board depending on the space and timing requirement. The choice of the most suitable of the proposed MLNS designs depends on the number of target filter taps and the requirements for the operational throughput, latency, and hardware availability.

Naturally, with the underlying neural net like structure, the MLNS filter
has the ability of adaptive filters. Our proposed MLNS (with adaptive learning-rate and structure modification) adaptive FIR filters have demonstrated improvements in the implementation of the classical FIR filters and the traditional adaptive FIR filters. An acceptable variable learning-rate and structure modification are achievable at a reasonable cost. Moreover, the MLNS for FIR filter is totally autonomous which does not require human in manipulating an FIR filter.
Semantic scene classification, robotic state recognition, and many other real-world applications involve multi-label classification with imbalanced data. In this paper, we address these problems by using an enrichment process in neural net training. The enrichment process can manage the imbalanced data and train the neural net with high classification accuracy. Experimental results on a robotic arm controller show that our method has better generalization performance than traditional neural net training in solving the multi-label and imbalanced data problems.
6.1 Introduction

Semantic scene classification, robotic state recognition, and many other real-world applications and optimizations involve multi-label classification, where a set of inputs is associated with multiple labels (outputs). For example, in semantic scene, a detected object (inputs) can belong to more than one object class (multiple labels). In robotic arm controller, a robot arm angle position may exist in more than one controller state. Unfortunately, most traditional classifiers [157, 32] can only handle single-label problems—, where a set of inputs is mapped to only one label, such as identification for one object class or the recognition of one controller state—or balanced data, where the classifier training data is uniformly distributed over the data space.

On the other hand, many classification problems also involve imbalanced data, where sampled data for the classifier training is non-uniformly distributed over the data space. The imbalanced data problem can take two distinct forms: either one class is under-sampled relatively to other classes, or it is over-sampled but too sparse in the sampling space. For example, a robotic arm controller uses the arm angle position and the sub-controller circuit gain to determine the controller states. The sampling of the controller has one state sparsely over-sampled relatively to other states due to the application design specification. Other applications, such as facial image associative memory [10] and adaptive self-configurable filters [8, 9], are also compatible with this approach.
Most learning algorithms, like traditional neural nets [16] and support vector machines [157], are designed for well-balanced data and do not work well on imbalanced data. While a traditional classifier can achieve very high accuracy by simply ignoring the minority samples, this is obviously undesirable because the minority samples may contain critical information which makes such a classifier useless in practice.

In this paper, we introduce a data enrichment process for neural net training to address the multi-label and imbalanced data problem. This enrichment process manipulates the imbalanced data into a smaller subset of more balanced data which will be used in the training of a neural net classifier. This subset is initialized through imbalanced data clustering and cluster re-sampling to obtain equally represented data in Euclidean space. The enrichment process iteratively updates the subset throughout the training phase by incrementally adding and removing data to maintain and improve classifier performance. The experimental outputs of the classifier show that our method has better generalization performance than the traditional classifier training.

This paper is structured as follows. In the following section, Section 6.2, some related works and our enrichment process overview are briefly introduced. In Section 6.3, the proposed enrichment processes, including initialization, updates, and termination, are described in detail. In Section 6.4, we perform experiments on robotic arm control problems and compare our methods with traditional classifiers.
6.2 Background

6.2.1 Related works

In the recent literature, some promising works have been reported with the development of multi-label classification algorithms, such as non-mutually exclusive class definition in Boutella [158], relevance feedback with supervised learning paradigm in Dorado [159], and multi-label conditional random field classification model in Ghamrawi [160]. These techniques have been shown to work with support vector machines but not for neural nets. Additionally, they assume a sparsely distribution of the training data set.

A number of approaches have also been proposed to address the imbalanced data problem. Examples include over-sampling of the minority class samples in Chawla [161] and adjusting the misclassification costs of the two classes in Japkowicz [162]. Nonetheless, these works have shown to be effective for Bayesian classifiers and decision tree systems—not neural nets.

Regarding works on data enrichment on neural net training, a few techniques to improve the training process have been proposed and proven to be effective under certain application constrains. Liu [163], Patil [164], and Salazar [165] propose the use of various training data clustering techniques to find cluster centroids using the cluster centroids as a training set for a single neural net. The results show good improvement; however, it is uncertain on how to select an appropriate number of clusters.
Another effort is to use multiple neural nets (ensemble) training on data clusters reported by Arslan [48], Cunningham [101], El-Gamal [166], Hartono [167], and Liu [168]. Each neural net is dedicated for training one data cluster. The results show better neural net performance in generalizing new inputs. Nevertheless, these approaches suffer from the use of multiple neural nets and their interpolation between the transition region of two adjacent clusters.

6.2.2 Overview of the Enrichment Process

In this work, we introduce the training data enrichment process (otherwise known as enrichment process), which is responsible for managing the balanced/imbalanced training data during the neural net classifier training. The enrichment process resamples the imbalanced data into a smaller subset of more balanced data. This subset is initialized through imbalanced data clustering and cluster re-sampling to obtain equally represented data in Euclidean space. The enrichment process iteratively updates the subset throughout the training phase by incrementally adding and removing data to maintain and improve classifier performance. The enrichment process operates solely during the neural net training phase—not during the normal operation.

The difference between our approach and other works are that we show the multi-label and imbalanced data problem can be addressed with a single neural net. Our result accuracy and capability are comparable to support vector machines, Bayesian classifiers, and decision tree systems mentioned earlier.
Additionally, we propose a data selection technique to improve neural net classifier training from the available imbalanced training set, which has not been addressed directly in existing works. Our approach avoids the assumption of sparsely distribution of the available training set as in [158, 159, 160]; it does not need data fusion techniques to integrate multiple neural net estimations (from ensemble) as in [101, 166, 167]. Though our approach uses a clustering technique, it does not require knowing the number of initial clusters in advance as in [163, 164, 165]; however, having that knowledge can be useful.

The enrichment process manages the training data within three steps:

1. *Enrichment Initialization* re-samples the available imbalanced training data or training set scope (T) to create a subset of more balanced training data or active training set (τ) for the first time neural net classifier training.

2. *Enrichment Update* incrementally adds and removes training data to/from the current active training set (τ) at the end of an enrichment training iteration. Additional training data are requested from the training set scope (T) to train the neural net. The modified active training set (τ) (after adding and removing of some data) remains a subset of the training set scope (T).

3. *Enrichment Termination* controls the enrichment process to iterate for a pre-defined number of enrichment training iterations.

We will describe the first two steps of managing the imbalanced data in Section 6.3. Then, we will illustrate their mechanisms and the benefits of the
6.3 Our approach

Our enrichment process to improve the performance of a single neural net is distinctly different from other existing works mentioned earlier. The enrichment process manipulates the available imbalanced training data or the training set scope \((\mathbb{T})\) by, first, systematically generating an initial subset of training data or an active training set \((\tau)\) by cluster re-sampling. Then, it incrementally modifies the subset during the neural net classifier training process. The data for neural net training is continuously added and removed with respect to the neural net bias to certain training data space (regions). The training data points which are relatively well-trained in the neural net should be dropped. On the other hand, each training data point which is relatively under trained should recruit more neighboring training data points. The incremental data modification process repeats to a pre-defined number of enrichment iterations. Figure 6-32 presents the pseudo code illustrating the enrichment process in three steps.

6.3.1 Enrichment Initialization

The enrichment process starts from the available imbalanced training data or the training set scope \((\mathbb{T})\) provided by a human investigator or a knowledge base. As for the first step, the enrichment process re-samples an initial subset training
**Step 1: Initialization**

a) Select a number, g.
b) Cluster the training set scope into g clusters.
c) Select equal number of data from each cluster as active training set.

**Step 2: Updates**

a) Train the neural net with the active training set.
b) Separate the active training set into 3 groups respecting to their energy errors.
c) Add neighbors for high−group and remove half of the low−group.

**Step 3: Termination**

a) Check if the enrichment process repeats for a pre−defined number of iterations.
b) If NO, repeat Step 2. Otherwise, stop.

Figure 6-32: Pseudo code for enrichment process

data or active training set (τ) from the training set scope (T) by applying a clustering algorithm based on Euclidean distance. The clustering technique (C) is initiated with an arbitrary initial number of clusters (g). If we have a priori knowledge of how the training set scope (T) is scattered in space, we can use the visual estimation for the initial number of clusters. Regardless of how the initial number of clusters is selected, the active training set (τ) will be balanced later on during the neural net training. These g clusters are solely used during the enrichment initialization; they will be discarded afterwards.

Once the training set scope (T) has been formed into g clusters, the enrichment process selects m training data from each cluster to produce the initial active training set (τ) where m is the least number of training data points in all g clusters. The selection technique (S) of m training data can be done by applying the same clustering algorithm (C) to the cluster with m sub-clusters. The training data points, which are closest (by Euclidean distance) to the sub-cluster centers,
represent the selected \( m \) points of the cluster. Hence, we can write the initializing process of the more balanced subset training data or the initial active training set (\( \tau \)) in the enrichment process as

\[
\tau = S(C(\mathbb{T}, g))
\]

About the computational complexity, the enrichment initialization performs the clustering algorithm (\( C \)) and the selection algorithm (\( S \)) on the training set scope (\( \mathbb{T} \)). According to Jain [169] and Berkhin [170], the clustering algorithm, based on k-mean Euclidean distance, has complexity of \( O(n \cdot \ln n \cdot r) \) order where \( n \) is the number of data (\( |\mathbb{T}| \)) and \( r \) is the information dimensionality of data (including input and output data). The selection algorithm applies the clustering algorithm onto all \( g \) clusters; this requires complexity \( g \cdot O(n \cdot \ln n \cdot \frac{r}{g}) = O(n \cdot \ln n \cdot r) \). The enrichment initialization only occurs once at the beginning of the enrichment training. Therefore, the enrichment initialization computational complexity is of the order

\[
O(n \cdot \ln n \cdot r)
\]

\[6.21\]

### 6.3.2 Enrichment Update

The training process will use the active training set (\( \tau \)) to train the neural net. Once the training is suspended after a pre-defined number of training iterations, the enrichment process re-examines the current training data by analyzing the
energy error $^4$ ($e_j$) of each training data ($t_j$) in the active training set ($\tau$). The energy error of the training data is normalized and compared against two preset thresholds ($e^h$, $e^l$ where $e^h > e^l$). These thresholds separate the subset training data ($\tau$) into three groups: high error ($\tau^h$), middle error ($\tau^m$), and low error ($\tau^l$) groups, in Equation 6-22.

$$\tau^h = \{ t_j | t_j \in \tau, e_j \geq e^h \}$$

$$\tau^m = \{ t_j | t_j \in \tau, e^h > e_j > e^l \}$$

$$\tau^l = \{ t_j | t_j \in \tau, e^l \geq e_j \}$$

With the three groups, we can concentrate neural net training on particular data which have not been very well-trained (indicated in high-error group). At the same time, we can ignore some data which are already well-trained (indicated in low-error group). The high-error group represents all current training data which are under trained in the neural net. The middle-error group is doing acceptable. On the contrary, the lower-error group represents all current training data for which the neural net is already well-trained or is biased.

The enrichment process ($E$) enriches the active training set ($\tau$) by adding (at most doubling) training data in the high-error group ($\tau^h$) and removing (at most half) training data in the lower-error group ($\tau^l$). The middle-error group is left unmodified. The enrichment process for adding training data finds the nearest neighbor (defined by Euclidean distance $^2$) for each training data in the high-error group ($\tau^h$). The nearest neighbors are added to the current training
data for the next enrichment training iteration.

On the other hand, the enrichment process for removing training data uses similar method of selecting \( m \) training data points from a cluster in the enrichment initialization. We need to select half number of training data in the low-error group \( (n = \left\lfloor \frac{|\tau_l|}{2} \right\rfloor) \). To select \( n \) training data points from the low-error group, we apply the same clustering algorithm (\( C \)) to the low-error group with \( n \) sub-clusters. The training data point, which are closest (by Euclidean distance\(^2\)) to the sub-cluster centers, will be kept for the next enrichment training iteration. The rest of the training data are removed from the current data for the next enrichment training iteration. Hence, we can represent the enrichment process of the training data as

\[
\tau = \tau_{i+1} = E(\tau_i, \{e_j\}, e_h, e_l)
\]

where \( i \) is the enrichment iteration index and \( j \) is training data index in the current active training set \( \tau_i \). The newly modified, more balanced training data or the new active training set \( (\tau = \tau_{i+1}) \) will be used in neural net training of the next enrichment iteration.

The computational complexity of the enrichment update depends on two main factors: the number of data \( (n = |\mathbb{T}|) \) and the number of enrichment iterations \( (p) \). The three-group separation \( (\tau^h, \tau^m, \tau^l) \) is a simple comparison against two preset threshold \( (e^h, e^l) \), which takes \( O(n) \) complexity. Then a neighbor for each high-group data points \( (\tau^h) \) is added to the active training set; the computation of finding an appropriate neighbor takes \( O(n) \) for one data point. Thus, the
total addition of the active training set requires \( O(n \cdot |\tau^h|) = O(n^2) \) complexity. Next step is to half the low-group \((\tau^l)\) by applying the clustering technique \((C)\); this needs \( O(|\tau^l| \cdot \ln |\tau^l| \cdot r) = O(n \cdot \ln n \cdot r) \) complexity. The enrichment update is repeated for a predefined number of enrichment iterations \((p)\). Therefore, the enrichment update computational complexity is of the order

\[
O(p \cdot n^2 \cdot r)
\]

6.3.3 Enrichment Termination

The enrichment process iterates through the enrichment update step for a pre-defined number of enrichment training iterations. The enrichment update uses the active training set \((\tau)\) to train the neural net using the evolutionary training technique in Tepvorachai [8] which is based on a modified back propagation. Similar to a conventional back propagation in Haykin [16], the evolutionary training repeats the training process for a pre-defined number of iterations. Like the neural net training, the longer the enrichment process repeats, the better the trained neural net can estimate the active training set. However, if we have a priori knowledge of how well the training neural net performs after a number of iterations, we can select the number of iterations that trains the neural net to a desired level of accuracy.
6.4 Experimentation

In this section, we are going to illustrate the results and the benefits of our enrichment process in neural net training. The scenario is the following. We face a great need to define a mapping between two input parameters and an output parameter. Such mapping is used in various applications, such as classification problems (i.e. image, fluorescence spectra, and gas dynamics) and control problems (i.e. robot arm, state space). The relationship between the two input parameters and the output is generally non-linear. Fitting and calculating an explicit mathematical 3-dimension function can be extremely tedious and computationally intensive. We choose, as an alternative, to use neural net as a candidate mapping function. In this example, we sampled data of a robotic arm controller in Figure 6-33 and plot them in 3 dimensions in Figure 6-34, where Angle and Gain are the two input parameters and State is the output parameter. Angle represents the current robot arm angle measured from a reference point (normalized from 0 to 1). Gain is an adaptive sub-controller unit input-output gain (normalized from 0 to 1). State refers to the expected system state of the robot arm.

Figure 6-34 shows the same sampled imbalanced control data of 100 sampling points. To calculate and fit all the sampling points with an explicit mathematical 3-dimensional function, we need to do trial-and-error on various modeling functions or require a certain high level of data modeling expertise. Alternatively, we can use the sampled points to train a neural net which also supports
Figure 6-33: Sampled imbalanced control data (multi-label) [note: blue “⊙” represents robot arm in state 0; red “□” represents state 1; and magenta “◊” represents state 2]

Figure 6-34: Same sampled imbalanced control data as in Figure 6-33 in 3-dimensional plot (multi-label)

a powerful interpolation function between the sampled points [32, 16]. However, using all 100 sampled points in conventional training can prove to deteriorate the training process (slow convergence rate and imbalanced data effect) [161, 162, 165, 48]. We will be using our training data enrichment process on a single neural net to speed up the training process with imbalanced data while
improving or maintaining the neural net accuracy in multi-label problems. Our results will be compared against a training process using all 100 sampled points on a single neural net and another training process with a randomly selected subset of the sampling points on a single neural net.

### 6.4.1 Setup

The following section describes the setups of the enrichment process. Besides the difference in the training data, all trainings share the following parameters. The neural net trainings are repeated for 80 iterations. The neural nets are setup with 2 inputs, 1 output, and 4 hidden neurons in 1 hidden layer. Note that, for the enrichment process, we do data enrichment update every 20 iterations. This means the enrichment initialization is at iteration 0; the enrichment updates are at iteration 20, 40, and 60. The reason we choose to do 4 enrichments at 20 iterations is to generate a fair comparison among the trainings to the total sum of 80 iterations (arbitrary number). The 4 enrichment splits the 80 iterations into even interval of 20 iterations. From our prior knowledge, the interval of 20 iterations is sufficient to train a neural net with the given architecture to a stable accuracy.

*Figure 6-35* illustrates the imbalanced training data (sampled control) or the training set scope (T) for neural net training where blue “•” marks all 100 training data (sampled control points), red “○” around a blue “•” means the selected training data will be used for initial enrichment neural net training, and dotted splines indicate initial enrichment clusters. The enrichment initialization...
divides the 100 training data into 3 initial clusters, from the visual inspection of
the training data scattering in space. As indicated by the dotted splines, each
cluster consists of 10, 42, and 48 training data points, respectively, from left to
right. The least number of training data in all 3 initial clusters \((m)\) is 10. Thus,
10 data points are selected from each cluster contributing to the initial more bal-
anced training data or the initial active training set \((\tau)\) (total of 30 points).

**Figure 6-35**: Initial enrichment active training set at enrichment iteration 1 [note: the dotted splines indicate 3 initial clusters.]

### 6.4.2 Results and Measurements

In the following section, we describe the neural nets evaluation after their train-
ings. During the enrichment training, we observe the trained neural net perfor-
ance. We measure neural net performance in two ways:

1. **Control effort accuracy** \((Ac)\) defined as the percentage of accepted effort in
   the sampled points. This measures how acceptable the neural net controller
is within a given tolerance level. Our robotic arm control process has the control state tolerance of 0.07 normalized units. If the estimated control state generated by the neural nets is within the tolerance range, it is considered acceptable. Otherwise, it will be rejected. The control effort accuracy can be mathematically defined as the percentage of acceptable control states ($accepted$) from all sample control states ($all$):

$$Ac = \frac{accepted}{all} \times 100\%$$  \hspace{1cm} 6-25

2. *Control effort energy error* ($E_e$) defined as the Euclidean distances (energy error$^4$) between the estimated control states and the sampled target control states. This measures how closely the neural net is trained to the given target regardless of the tolerance. The smaller the energy error, the better the neural net performance is. The control effort energy error can be mathematically defined as:

$$E_e = \sqrt{\sum_j (t_j - o_j)^2}$$  \hspace{1cm} 6-26

where $j$ is all sampled control points.

We measure the neural net’s performance after enrichment iteration. The total number of enrichment iterations for this example is 4. Table 6-9 shows the two neural net performance measurements at the end of iterations. Figure 6-36 illustrates the last enrichment update training data where blue “•” marks all 100 training data (sampled control points), red “○” around a blue “•” means the
selected training data for enrichment iteration 4. We show Figure 6-36 as the enriched data selection comparable to the initial data selection in Figure 6-35.

<table>
<thead>
<tr>
<th>Iterations</th>
<th>Accuracy</th>
<th>Energy Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>27.00%</td>
<td>2.0759</td>
</tr>
<tr>
<td>2</td>
<td>87.00%</td>
<td>0.4353</td>
</tr>
<tr>
<td>3</td>
<td>89.00%</td>
<td>0.3958</td>
</tr>
<tr>
<td>4</td>
<td>98.00%</td>
<td>0.3104</td>
</tr>
</tbody>
</table>

Table 6-9: Neural net performance over enrichment training process

During the enrichment training process, we observe the number of training data points being clustered into three error groups: high-error, middle-error, and low-error groups, as mentioned in Subsection 6.3.2. The number of training data points in the three group assignment is illustrated in Figure 6-37. The first enrichment iteration (iteration 1) is the enrichment initialization. The enrichment update occurs after enrichment iteration at 2, 3, and 4.

In addition to the three error group observation, a few training data have been added and removed to/from the active training set due to the enrichment process.
Figure 6-37: Number of training data points in high-error, middle-error, and low-error groups

update. The number of training data points being added and removed to/from the active training set is shown in Figure 6-38. The first enrichment iteration (iteration 1) is the enrichment initialization; there is no enrichment update (no training data being added or removed).

Figure 6-38: Number of training data points being added and removed
From the result in Figure 6-37 (magenta “△” marker) and Figure 6-38, we observe the number of training data points during the training process. Enrichment initialization produces training data with 30 points (1st enrichment iteration); enrichment updates (2nd – 4th iterations) have 26, 27, and 29 data points, respectively.

Moreover, we measure how the active training set (τ) from the enrichment initialization and the enrichment update distributed over the training data space. We measure the distribution by the training data variance. Recall that the variance (ρ^2) of 1-dimensional data is defined as the average of the Euclidean distance square from data mean in Equation 6-27 [171].

\[
\rho^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2 \tag{6-27}
\]

We can calculate the variance of our 3-dimensional data by replacing the 1-dimensional Euclidean distance square by 3-dimensional Euclidean distance square. Hence, we obtain our training data distribution (variance) measurement in Equation 6-28.

\[
\rho^2 = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{3} (x_{i,j} - \bar{x}_j)^2 \tag{6-28}
\]

The training data distribution and data mean (average)—as a vector of the low-error, middle-error, and high-error groups, respectively,—for the active training set (τ) in enrichment iteration is shown in Table 6-10.

Besides the neural net training with the enrichment process, we created two comparison neural net trainings. We refer to the neural net with enrichment training as “enriched” neural net. One of the comparison trainings, indicated
Table 6-10: Active training set distribution and training data mean (average) as “all-data” neural net, refers to a neural net training with all sampled imbalanced control data from Figure 6-34. Another comparison training, indicated as “random” neural net, refers to a neural net training with some randomly selected training points from the sampled imbalanced control data in Figure 6-34. The “random” neural net uses 29 randomly selected training data which is about the same number of training data points that “enriched” neural net uses. The comparison of neural net performance at iteration 80 is shown in Table 6-11. The Accuracy, Energy Error, and training data Distribution are defined in Equation 6-25, 6-26, and 6-28.

<table>
<thead>
<tr>
<th>Iterations</th>
<th>Distribution</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5311</td>
<td>(0.3700, 0.4183, 0.7303)</td>
</tr>
<tr>
<td>2</td>
<td>0.4734</td>
<td>(0.5019, 0.4827, 0.9591)</td>
</tr>
<tr>
<td>3</td>
<td>0.6185</td>
<td>(0.4056, 0.4278, 0.7748)</td>
</tr>
<tr>
<td>4</td>
<td>0.4958</td>
<td>(0.4500, 0.4931, 0.8940)</td>
</tr>
</tbody>
</table>

Table 6-11: Comparison neural net performance at iteration 80 [* is the average distribution over the enrichment process in Table 6-10]

6.4.3 Analysis and Discussion

From Figure 6-37, the “enriched” neural net training (with more balanced data about 26-30 points) uses much less training data than the “all-data” training (with
100 imbalanced training data points). The “enriched” and “random” (29 randomly selected training data points) neural net use about one-third of the available imbalanced data.

Regardless of the number of training data points for neural net training, the measurement in Table 6-11 shows the “enriched” neural net has higher accuracy (98.00%) than the “all-data” (75.00%) and “random” (72.00%) neural net classifier; at the same time, it has lower energy error (0.3104) than the others (1.2655 and 1.4792). Additionally, the training data distributions of “enriched” and “random” training (0.5297 and 0.4958) are almost indistinguishable; however, they are differentiated from the “all-data” distribution (0.3903).

Additionally, with regards to the result in Table 6-9, the enrichment process has incrementally improved the neural net classifier performance. These results demonstrate that the enrichment process contributes significant improvement to the neural net classifier training process on multi-label imbalanced training data. The enrichment training loop can also be extended into higher information dimension neural net training (including input and output data).
Chapter 7

Facial Image Associative Memory Model

Facial image associative memory takes a facial input image and returns associated faces pre-embedded in memory. This work proposes a three-phase implementation process: a) sensory pre-processing, b) information interfusion, and c) association with existing faces. This work reports on the simulation and performance of the proposed first phase, sensory pre-processing, based on multiple neural net structures to translate image sensory pre-processing into transformed information. The multi-network structure is tested by 46 faces of 21 individuals. The result shows the first phase can produce acceptable associations at 89.1% of all test faces with less than 0.02 energy error at above 0.60 facial image pixel-based correlation (closeness).
7.1 Introduction and Motivation

Associative memory is a content addressable memory which takes an input data key and returns associated data items that are embedded in memory. A classical associative memory [172] is a system that stores the mapping of specific input representation to output representations [173]. The classical associative memory can be extended into the image associative memory which has particular potential benefit in surveillance, reconnaissance, and biological-inspired cognition modeling [156].

Many existing works have attempted to model facial image associative memories with physical/optical devices [174], image pre-processing [175, 176], and software modeling [177, 178]. The attempts include optical network device implementations, various image filtering and manipulation techniques, recurrent neural net model, and direct imagery pixel-based operations. These methods are intensive computations in real-time by analyzing pixel-based information or considering the entire input image for correlation with existing images in memory which consumes much memory space and computation time.

In our approach, the facial image associative memory is implemented by a three-phase process: 1) sensory pre-process multiple feature of the input facial image, 2) interfuse the multiple sensory information, and 3) associate the interfused information with existing faces embedded in memory. The three-phase process employs neural nets [16] to approximate the imprecise pre-processed
sensory data, multiple data interfusion, and information associations with ex-
isting data.

This facial image associative memory model does not consider the entire
face correlation or analyze whole face by pixels. Our model decomposes the in-
put facial image into sensory multiple component keys\(^5\) (or image keys) and
produces association levels for all faces stored in memory. A set of faces with ac-
ceptable association levels is recalled as the associated faces for the given input
facial image.

The following section, Section 7.2, presents a brief background informa-
tion of the classical associative memory, related works, and our contributions.
Section 7.3 describes the memory model design proposed of the three phases
and the augmented face detection. Achieved simulation results are illustrated in
Section 7.4. Section 7.5 discusses some limitations and potentials.

### 7.2 Background

#### 7.2.1 Classical Associative Memory

Associative memory is usually referred to a type of computer memory or an as-
pект of human memory. This work focuses on associative memory as a computer
memory model categorized into a type of content addressable memory (CAM)
[173, 172]. A CAM is designed in a way that the user supplies a data key and the
CAM looks up the memory. If the data key is stored anywhere in the memory, it
returns the associated data item(s).

Association refers to the process of bringing objects or concepts together in memory. The process signifies a relationship between two or more objects or variables. A relation can be described as the mapping of specific input representation to specific output representation. The CAM model suggests that associative memory associates the data key with a list of data items. Therefore, associative memory represents relationships among data keys and data items stored in the memory. An abstract model of associative memory is illustrated in Figure 7-39.

Figure 7-39: Abstract associative memory model

The relationship includes 1-to-1, many-to-1, 1-to-many, or many-to-many relationships. With wide variations of data component keys and many relations among objects, imprecision is introduced as the strength of associations between associated object pairs. The imprecision maybe captured with statistical models of closeness or object representations (hashing functions) as in [173]. This work focuses on the 1-to-many relationship characteristic of associative memory.
7.2.2 Related Works

Many researchers have investigated in facial image associative memory. Neifeld and Psaltis demonstrated a closed-loop optical associative memory based on optical disk incorporating image correlation to compute the best association in a shift-invariant fashion [174]. When presented with a partial or noisy version of one of the images stored on the optical disk, the optical system evolves to a stable state in which those stored images that best match the inputs are temporally locked in the loop. Cortadellas and Amat proposed a facial image associative memory based on silhouettes representation [175]. However, the processing is impossible to distinguish two objects with similar silhouettes but different textures, colors, and other features. In a more recent research, Zhang et al presented a face recognition system using a Gabor wavelet associative memory [176]. The Gabor wavelet associative memory has shown superiority and demonstrated very high performance comparing favorably to some recent face recognition methods against varying illumination conditions. All these efforts have been based on the matching of whole image correlations which need to examine the entire image content.

Skarbek introduced the concept of pixel based neural net (PNN) which does grayscale facial image auto-association using information at pixel level [178]. Comparing to k-NN [179], the PNN uses much less memory space per one person while performances for both techniques on average are very close. The PNN,
however, is designed for auto-association application which might not be appropriate for hetero-association. O’Keefe and Austin presented an associative memory neural net to identify features in an image and recall associations between features and the objects which they comprise [177]. The network is operating on black-and-white images and limited to simple object recognition based on Generalized Hough Transform.

7.2.3 Our Contributions

This work presents a facial image associative memory model which relates an input facial image with a set of faces already stored in memory. The facial image associative memory is a three-phase implementation process: a) imprecise sensory pre-processing of multiple features, b) multiple information interfusion, and c) association with existing faces. The paper contributes to the design, experimental simulation (training/testing), and analysis of the proposed first phase of this facial image associative memory model. These three phases are embedded with multiple neural net structures to approximate the face associations.

The facial image associative memory model avoids unnecessary intensive computation by analyzing only sub image features (image keys or component keys) extracted from the input facial image for association. It avoids the correlation computation by transforming the sub image features into smaller pieces of information (transforms), for more convenient to manipulate and consume less memory space and computation time. Moreover, the memory model does
not use recurrent neural nets to recall faces; instead, it stores all faces and their transforms in look-up tables for quick recall. [175, 174, 177, 178, 176]

This work leverages existing face detection work. Face detection is to separate human faces within an image for the sensory pre-processing phase which will be briefly discussed in Subsection 7.3.3.

7.3 Facial Image Associative Memory Design

Figure 7-40 presents our proposed three-phase process for the facial image associative memory model extended from Figure 7-39. A data item may be represented with highly detail information such as sound or image. This work focuses on associative memory for facial images (still pictures). The facial image keys are pre-processed by multiple sensors; the multiple-sensory information is interfused into one; then, the associators decide which face items are related (or associated) with the facial image keys.

![Figure 7-40: Facial image associative memory model three-phase process](image)

The multiple sensors in this associative memory model present potential similarity with natural recognition systems such as face recognition in human. When people look at a face, they speculate on the eyes, the nose, the mouth,
and other facial marks and draw on their experience whose face it belongs to. The multiple-sensor scheme acts similarly to the speculations on those facial features. One visual sensor draws on the eyes; another considers the nose; one looks at the mouth. Then, the multiple pieces of information from the sensors are merged and fused into one item. This fused information is associated with some known faces in memory. The sensors need not belong to the same sensing methods (all visual or all audio); they can be mixed and matched depending on applications, such as audio, visual, pressure, and chemical for surveillance application [156].

7.3.1 Sensory Pre-processing

The sensory pre-processing of input facial image component key may proceed with multiple sensors. Digital images consist of pixel information; however, operations by pixels are not computationally efficient. Each sensor performs specific pre-processing task to extract only necessary information from the original facial image key such as its Fourier transform or wavelet. The extracted information is then converted into smaller transformed information (transform) which will be used in later phases. A sensory pre-processing takes one facial image component key and produces one transform as its output. The simultaneous sensory pre-processing operations are illustrated in Figure 7-41.
The pre-processing boxes, in Figure 7-41, take sub image features (component keys or image key) located in the original input facial image and extract information from them. The extracted information may be different in data structures subject to the algorithm each specific pre-processing uses. For example, a Fourier pre-processing generates a Fourier image with the same image dimension as its input image component; a wavelet pre-processing generates a wavelet coefficient vector of its input image component. The extracted information is then converted into transforms by the transformers, transformer boxes. Each transformer is responsible for one pre-processing unit; hence it is pre-processing specific, as shown in Figure 7-41 and Definition 1.

**Definition 1 (Pre-processing Transformer).**

Let $A_i$ be extracted information produced from pre-processing box of sensor $i$ in Figure 7-41, where $i = 1, 2, \ldots, s$.

Let $f_i$ be a mapping from extracted information from sensor $i$ to transform, where $i = 1, 2, \ldots, s$.

Define $a_i = f_i(A_i)$ as transformers.
The vectors $a_i$ are the resulting transforms produced by the transformer $f_i$.

The facial image component keys maybe analyzed and extracted by other existing techniques such as principal components analysis, edge/corner detection, invariant feature transform, and statistical wavelet analysis. Yuan et al developed an image feature extraction method based on the statistical analysis in the wavelet domain for content-based image retrieval [180]. The method results indicate the composed indexing feature space through the statistical approach is very effective in representing image features and provides a high retrieval rate. Hyvarinen et al reported on how sparse coding can be used to extract wavelet-like feature from image data [103]. Methods based on sparse coding have the important benefit over wavelet methods that the features are determined solely by the statistical properties of the data, while the wavelet transformation relies heavily on certain abstract mathematical properties that may be only weakly related to the properties of the natural data.

The transformation process is preferential and imprecise with respect to the relationship between facial image component keys and face items stored in memory. Traditional mathematical equations or programming algorithms for modeling the transformation can be highly complex, which is not suitable; hence, this work proposes the use of neural net (NN) [16] for preferential transformation. The new sensory pre-processing phase model is modified as demonstrated in Figure 7-42.
7.3.2 Interfusion and Association Phases

Each sensor produces one transform. The transforms of the multiple sensors are collected and interfused. In Figure 7-43 interfusion box, the interfusion merges all the transforms into one interfused transform. The transform interfusion process usually demands statistical preference. Many statistics have been studied, such as mean, median, variance, density distribution, arithmetic coding, LZW, Hoffmann, etc [181, 182, 183]. The traditional statistical processes are normalized and usually unbiased. Unfortunately, the facial imaging application in this work contains bias information concerning the relationship between the facial image component keys and the stored face items in memory. The possibility for bias transform interfusion among the multiple-sensory information will be further investigated.

The interfused transforms are then compared against existing face transforms in memory using associator similarity measurements to determine their
association levels. For each of the face items stored in the memory, the comparison between the stored face transform and the interfused transform is calculated by an associator, shown in Figure 7-43 association box. Similarity measures may refer to statistical tests like closeness measure, distance measure, Bayesian statistical similarity, or graph matching [184, 185, 186, 187]. The measurement is possibly Euclidean distance, Manhaton distance, Hamming distance, Hausdorff distance, or similarity test. To measure association level statistically, a mathematical procedure is necessary to capture a pre-defined association level. However, in real application, association is not well-defined or not known. The possibility for ill-defined/undefined association level between two pieces of information will be further investigated. For example, two images (such as an apple and an orange) which have low image correlation might have high association level in a given context relationship (they are fruits); we are interested in the later association level.

7.3.3 Augmented Face Detection

Figure 7-43 illustrates the facial image associative memory model after combining the proposed first phase process with the other two phases described above, in Subsection 7.3.2. The face detector is presented primarily for facial imaging application. This work focuses on facial image association memory. Faces in the input image need to be located before feeding them to the three-phase process.

JNI2OpenCV
Figure 7-43: Image associative memory model in large scale

The first half of the face detector is already implemented in a Java package called JNI2OpenCV, freely available online. JNI2OpenCV is a Java interface to native language in C/C++ OpenCV library for computer vision researches and applications supported by Intel Technology & Research under open source licensing.

The face detector, JNI2OpenCV, reads in a 24-bit RGB image and returns all potential face locations within the image regardless of the background. Other face detections may be a number of other techniques presented in [188, 189, 190, 191, 192, 193].

**Face Mask**

JNI2OpenCV supports only frontal, upright faces (no in-plane or out-of-plane rotations). It cannot locate any positions of facial features such as eyes, nose, and mouth. The potential positions of the facial features may be extracted with a face mask. Specifically for this application, we use a face mask to locate eyes, nose, and mouth positions from the extracted faces. The face mask is constructed from relative positions of the eyes, the nose, and the mouth with respect to the upper-right corner of the face with 1.0 being the full width/height. A set of
sample faces (training set) were presented and the relative positions were manually collected. The average positions over all collected samples define the face mask. The relative eyes, nose, and mouth positions are shown in Table 7-12.

<table>
<thead>
<tr>
<th></th>
<th>Relative Positions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>eyes</td>
<td>0.176</td>
<td>0.240</td>
</tr>
<tr>
<td>nose</td>
<td>0.381</td>
<td>0.420</td>
</tr>
<tr>
<td>mouth</td>
<td>0.307</td>
<td>0.709</td>
</tr>
</tbody>
</table>

Table 7-12: Face mask relative positions

7.4 Experimental Simulation Results

We train the proposed first phase of our facial image associative memory model with augmented face detection. We are interested in three facial features for association: eyes, nose, and mouth, as defined in face mask. The sensory preprocessing phase has three sensor units corresponding to the eyes, the nose, and the mouth. Each facial sup image feature is passed to each corresponding sensor unit. Each sensory pre-processing unit, shown in Figure 7-42, takes facial sub image feature from the face mask as inputs. The pre-processed results are inputs to the transformer neural net structure.

The transformer network structure consists of multiple neural nets; one network for one pre-processing unit, as shown in Figure 7-42. In this experiment, we use three neural nets; one for the eyes, one for the nose, and one for the mouth pre-processing transforms. Each network consists of an input layer, an output.
layer, and a hidden neuron layer with 9 hidden neurons. At each layer, a bias term is permitted. The networks are complemented with separated weight updates and separated adaptive learning rate adjustments. The goal for the transformer multi-network structure is to transform the pre-processed results, from the pre-processing boxes, to the associated face transform as defined in the training set with the least energy error\(^4\). The complementary weight updates attempt to minimize this energy error.

We train the transformer multi-network structure with 20 faces from 10 individuals. The faces for this training set are selected from CMU NNFaceDetect face database [194] and BioID face database [195]. We use the face detector and implement sensory pre-processing and neural net structure with Java 1.5 SDK on Java Virtual Machine 5.0 with 1GB virtual memory. The JVM runs on Sun-Blade1000 with 1GB of RAM. The training process simulates for 33,000 training iterations; one training iteration exposes the transformer multi-network structure to each association face pair. The total energy error of each iteration is plotted against the number of iterations as illustrated in Figure 7-44 (a) in linear scale and (b) in log-log scale.

After the training process is completed, the trained facial image associative memory is tested against 46 faces from 21 individuals. The images for this testing set are also selected from CMU NNFaceDetect face database [194] and BioID face database [195]. The result shows every face in the testing set produces
transforms with less than 0.04 energy error\(^4\), Figure 7-45, even though their image correlations\(^6\) of the associated pairs are between 0.60 to 1.00, Figure 7-46.

For the testing image set (46 faces from 21 individuals), we set an acceptable association threshold at 0.02 energy error. This means, for a facial image pair to be considered associated, the sensory pre-processing must produce transforms that have energy error less than the threshold. Out of 46 faces, the associations are acceptable for 41 face pairs with their image correlations between 0.60 to 1.00. Figure 7-45 illustrates the energy error result of each testing image produced when they are compared to their target images, and Figure 7-46 illustrates the facial image correlation between the input-target pair faces.
7.5 Discussion and Analysis

The rejected five face pairs, indicated in Figure 7-45 and 7-46 with arrows and dotted boxes, have energy error greater than the pre-set threshold level 0.02. They are not considered as valid association results. However, their image correlations
are between 0.60 to 1.00 with 1.00 being a perfect match and 0.60 does not spatially match well. These image correlations of the two groups (accepted and rejected) show that our facial image association memory model does not depend on the overall facial image correlations to determine association.

The proposed three-phase process, in Figure 7-43, demonstrates $s$ imaging sensors with $p$ transform interfusers on $n$ images previously stored in the memory. The total number of images is $n$, which are stored in a simple 2-tuple table, is usually much larger than the number of interfusers $p$ and the number of sensors $s$. In typical sensory applications, the number of sensors should be at minimum to reduce costs and maintenance. Also, the number of sensors $s$ should be less than the number of interfuser $p$ needed to interfuse the sensory transforms.

These examples of facial image association model are limited to pictures of real human faces. For example, JNI2OpenCV does not support cartoon character faces. The current facial image associative memory can be extended to the cartoon character if cartoon face detection is available to replace JNI2OpenCV. Usually, the image association model works exclusively on a limited image set such as human face image set or cartoon face image set. More generally, we can extend this image association memory model onto terrain association assuming that terrain detection is available [156].
Chapter 8

Conclusion and Future Works

8.1 Discussions and Remarks

8.1.1 Remarks on Neural Net Ensemble

Evolutionary process learning employs a neural net-based training and mapping approach between input and output representations via an evolutionary learning algorithm. Neural nets can be used as non-linear statistical data modeling tools. They can be used to model complex relationships between inputs and outputs. In our approach, neural nets are employed at all three significant processing phases, mentioned earlier in Chapter 3, in which the relationships between their inputs and outputs are difficult to define explicitly. Neural nets can potentially support through training: (a) complex non-predefined relationships, in sensory processing and feature extractor phase); (b) bias correlation, in feature interfusor phase; (c) preferential association, in cognitive recall. These capabilities make neural
nets very desirable.

Other approaches not based on neural nets have been developed. In information recognition application, the template matching [196, 197, 198] is a technique for finding small parts of data which match a priori template. However, the template in this application may not be known beforehand or may be difficult to generate. The Hidden Markov model [198, 199, 200, 95] is another statistical model in which the system being modeled is assumed to be a Markov process (a stochastic process that has a Markov property). In this application, the system architecture cannot guarantee the Markov property. Bayesian networks (or belief networks) [201, 198, 202] is a probabilistic graphic model that represents a set of variables and their probabilistic independencies. The system architecture proposed is too complex for an independency probabilistic graphic model. Therefore, the neural net model is the most suitable candidate for our purposes.

All three phases of cognitive processing use an evolutionary neural net training and learning scheme to perform their internal processing. Evolutionary learning supports offline training during traditional training sessions. It may also possibly support online training during real time application.

The reasons behind using the neural net ensemble approach in cognitive processing is two fold:

1. The classification errors of the individual members of the ensemble cancel out to some degree when their outputs are combined. Accordingly, higher classification is achieved.
2. The mapping from the input space to the output space is not approximated by single sensory information but by several sensors, where each neural net can focus on a particular sensor in the input environmental information space. As such, a proper mapping between the input-output spaces of the cognitive processing is constructed which will in turn leads to a better generalization. Moreover, the selection and training of a suitable neural net architecture is simplified.

Ensembles are desirable due to the basic fact that selection of the weights is an optimization problem with many local minima. All global optimization methods in the face of many local minima yield “sub-optimal” parameters (weights) which differ greatly from one run of the algorithm to the next, i.e., show a great deal of randomness stemming from different initial points (initial weights) and sequencing of the training examples. This randomness tends to differentiate the errors of the networks, so that the networks will be making errors on different subsets of input space.

8.1.2 Evolution and Machine Learning

The approaches of computational intelligence should be employed toward the goal of modeling evolutionary systems. Evolutionary computation, neural networks, fuzzy system, swarm intelligence, and expert systems can all contribute to a varying extent in the evolutionary systems. It is thought that the human brain uses multiple techniques to both formulate and cross-check results. The
approaches are complementary; the evolutionary system attempts to possibly combine and implement the four approaches in some combinations. Expert inference rules can be generated through computational neural network or production rules from statistical learning. Evolutionary computation can explore population based mechanic to find optimal solution. While the system environment keeps changing, the evolutionary system can track the changes with fuzzy systems. The integration is seen as promising and perhaps necessary for true artificial intelligence.

8.1.3 Information Integration

The use of sensory data from a range of multiple sensors is to automatically extract the maximum amount of information possible about the sensed environment under all operating conditions. Increased performance, reliability, data rates, and autonomy, coupled with increased complexity, and diverse uncertain operating environments, requires the automated intelligent combination of data from multiple sensors to derive less ambiguous/uncertain information about the desired state. This makes the system less vulnerable to failures of a single component and generally provides more accurate information. The basics of information fusion [108] is: the combining or merging of data from different sources in a manner that provides extra information and/or better quality than any of the single sources involved. In this direction many investigators have attempted to solve the fusion problem through different techniques [110, 109, 203, 204].
The main potential advantage of data fusion is that by integrating complementary information from different sources, the uncertainty of fused information is minimized and the fused performance can be improved.

Evolutionary systems attempt to achieve human intelligence in an evolutionary development process through machine learning and information fusion technology. The concept of system evolution and information cognition can be applied to all evolutionary systems including biological systems, evolvable filter, image feature extraction, collaborative data, preference learner, space explorer, and associative memory. Such integration will be investigated specifically in each application.

8.2 Summary of Research Contributions

The main innovation of our method is a cognitive information processing scheme consisting of three phases:

1. multisensory signal filtering into features,

2. fusion of the sensory features into signature associated with a target, and

3. cognitive type of matching of the input sensory signatures to similar target signatures pre-stored in the memory or in the knowledge base.

The key element of our method is the use of a novel evolutionary technique for offline learning and training of realistic target signatures under several experimental scenarios. The learning includes coefficient training, structural evolution, and
enrichment process.

The proposed evolutionary learning (training scheme) differs from the traditional training [16, 49] in an evolutionary aspect, that is, the growth/shrinkage of the neural net structure and the training data themselves progressively evolve. Thus the key advantage of our evolutionary learning scheme is based on two processes, the neural net structural evolution and the training data enrichment, which progressively iterate to produce enhanced training.

A key issue of our approach is how to analyze and evaluate multiple sensory data concerning the target in real time. Patterns of diverse sensory data could be weighted to provide real time responses. Sensory patterns could be recognized by adaptive techniques and systems that have been trained, offline, based on data from previous experimental scenarios. A key property of the system is the adaptation, training and learning capability of the system based on a cognitive processing scheme to provide responses in real time.

The cognitive processing on evolutionary platform contributes to the development of retargetable information processing applications with a high degree of knowledge and skills relating to

- Define and develop cognitive processing based on evolutionary platform that address both design and performance aspects of information cognitive model.

- Define and develop evolutionary platforms for retargetable information processing applications that address both design and performance aspects of
evolutionary system framework. It accommodates the growing of past experience on dynamic information centric strategies.

- Develop a neural net learning scheme (coefficient training, structural evolution, enrichment process) which is able to determine the number of neural net layers, the number of neurons in each layer, and the weights of the connections between neurons in different layers.

- Define the representation of application objectives and sensory suite to guide the development of cognitive processing on evolutionary platform for potential retargetable information processing applications.

- Develop potential retargetable information processing application test scenarios for cognitive processing on evolutionary platform.

- Demonstrate the cognitive processing on evolutionary platform on selected potential retargetable information processing applications as a proof-of-concept.

The material supports the establishment of a knowledge-base that allows the engineers and researchers to recognize the characteristics of the various operators and the strategies of computational intelligence, evolutionary technique, information cognitive model, and their specific utility to retargetable information processing applications. This approach should lead to the ability to construct appropriate cognitive processing schemes which is embedded in the design of evolutionary platform.
The cognitive processing on evolutionary platform introduces a paradigm shift from static fixed architecture and specific software optimization to retargetable information processing schemes, application software, and hardware implementation. Our approach is to demonstrate the motives and goals, to investigate the research questions. To support the claims, we develop conceptual system framework, sound argument, and formal analysis supported by existing literature references, figure illustrations and mathematical equations where possible.

The cognitive processing on evolutionary platform and the methodology developed in this work for retargetable information processing applications based on adaptive behavior have been demonstrated to be successful in producing automated information cognitive modeling based on the intended application and environment. The cognitive processing on evolutionary platform is applied to adaptive finite-impulse response filter in a software simulation level and in an FPGA hardware implementation level. The cognitive processing is also applied to facial image associative memory, resulting in successful realistic face recall. This methodology produces adaptive information cognitive model whose training time, success rate, and memory capacity exceeds existing work expectations.
8.3 Limitations

Nonetheless, there is an important drawback to the evolutionary computing. All evolutionary and adaptive processes in biological system or computational system can take as long as it needs to produce a perfect result [205, 5, 16]. In computational system, sometimes it takes two minutes to approximate a simple system, but it can take too long to approximate other highly complex system. This issue is studied in the Perceptron [16] project a few decades ago. It is suggested that a certain level of error is necessary for the evolutionary and adaptation process to halt and return an appropriate result.

Though the cognitive processing on evolutionary platform can serve as a strong predictive model for finding complex relationships, it will not produce explicit relations. Therefore, the model obtained gives little insight into the underlying mechanisms of the problems.

8.4 Publications

**Self-Configurable Neural Network Processor for FIR Filter Applications [8]:**

A self-configurable system is one that is designed primarily for the purpose of reconfigurable control and adaptive signal processing. It evolves by restructures and readjustments back and forth which can track the environment and the system variation in time. Processing methods and application areas include but not limited to transmission enhancement such
as filtering, equalization, and noise cancellation. The performance of our proposed self-configurable neural network processor for finite impulse response (FIR) filter are compared with those of the classical FIR filters and the traditional adaptive FIR filters. The neural network processor is an autonomous system which does not need human design knowledge of the FIR filter.

**Configurable FIR Filter Scheme based on an Adaptive Multilayer Network Structure [9]:** In this work, we present a design technique of configurable FIR filter architecture based on neural network like (multilayer network) structure. This architecture is a generalization of the configurable adaptive FIR filters and can be implemented on an FPGA. The design is based on a combination of pipelining and folding schemes at the multiplication-addition component level, network cell level, and network layer level. The proposed configurable multilayer network technique can reduce latency and hardware requirements while increasing the throughput of the filter. The weighted input connections, the network cells, and the number of network layers are configurable to fit filter design requirements. The configurable pipelining/folding scheme and the parameter setting characterize the proposed FIR filter architecture, FPGA space, and operation timing requirements. The configurable architecture is compared with several traditional FIR filter structures regarding hardware and time complexity. The potential applications of the proposed architecture are also discussed with respect to traditional
adaptive FIR filter performance.

**Facial Image Associative Memory Model** [10]: Facial image associative memory takes a facial input image and returns associated faces pre-embedded in memory. This work proposes a three-phase implementation process: a) sensory pre-processing, b) information interfusion, and c) association with existing faces. This work reports on the simulation and performance of the proposed first phase, sensory pre-processing, based on multiple neural network structures to translate image sensory pre-processing into transformed information. The multi-network structure is tested by 46 faces of 21 individuals. The result shows the first phase can produce acceptable associations at 89.1% of all test faces with less than 0.02 energy error at above 0.60 facial image pixel-based correlation (closeness).

**Multi-Label Imbalanced Data Enrichment Process in Neural Net Classifier Training** [11]: Semantic scene classification, robotic state recognition, and many other real-world applications involve multi-label classification with imbalanced data. In this paper, we address these problems by using an enrichment process in neural net training. The enrichment process can manage the imbalanced data and train the neural net with high classification accuracy. Experimental results on a robotic arm controller show that our method has better generalization performance than traditional neural net training in solving the multi-label and imbalanced data problems.

**Face Recognition using a Cognitive Processing Model** [12]: In the conventional
eigenface method, the principle component analysis (PCA) algorithm associates the Eigen vectors with the changes in illumination. In this paper, we propose an improvement of facial image association for face recognition using a cognitive processing model. This method is based on the notion of multiple-phase associative memory. The Essex face database is used to verify our model for facial image recognition and compare the results of face recognition with conventional eigenface method. The simulation results show that the proposed cognitive processing model approach results in better performance than that of the conventional eigenface approach; while the computational complexity remains of the same magnitude as that of the eigenface method.

8.5 Future Works

Although the assumptions and methodologies of cognitive processing on evolutionary platform have been very successful for retargetable information processing applications, there remains much work to be done. In addition to the current applications of the cognitive processing on evolutionary platform (adaptive filter and associative memory), we may wish to extend, recognize and reproduce more complex cognitive model, such as animal cognitive, child cognitive or human cognitive behaviors.

Currently the cognitive processing on evolutionary platform is taken to include associative memory for facial information cognitive model with sensory
processing, signature fusion, and cognitive recall. However, it is possible that there may be different behaviors triggered by different information which may be observed as additional model inputs or sub-components. The inclusion of these possibilities would result in a cognitive processing on evolutionary platform that is less rigidly structured than the current architecture. Therefore, it may be applicable to a wilder range of applications. However, that increase in input information and information processors may also increase the difficulty in identifying or parameterizing behaviors in some new information cognitive models.

Finally, the cognitive processing for information cognitive model in facial image associative memory is only test case with realistic information. The cognitive processing and the methodologies developed in this work have yet to be tested on actual industrial systems. In the future, it should be possible to design actual commercial systems to collect and classify information from the real industrial environment. The methodologies developed in this work might then be used to observe real-time information and produce cognitive actions with respect to the environment on the cognitive system.

The future works are summarized as follows:

- In this thesis, the proposed cognitive processing has been demonstrated in a very controlled environment. It has not been tested on an actual system environment, such as on commercial systems. We may investigate on how to efficiently deploy the cognitive processing in real industrial environment setup with real-time information.
• We may further investigate the current facial image recognition application aiming to identify other types of sensory information that may improve the recognition accuracy. The different features on human faces may be observed as additional sensory information to the current cognitive processing model. These observations may lead to improvements in other related applications.

• We may extend the proposed cognitive information processing to more complex problems, such as car (rare-view) recognition, scenery recognition, footprint recognition and literature recognition. More general cognitive problems may include animal and human cognitive behaviors.
Appendix A

End Notes

1 Available training data is all the training sets the evolutionary learning receives from human operators to train the cognitive processing on the evolutionary platform. The available training data depends upon the intended application which may be multi-labeled and/or unbalanced.

2 Other clustering techniques and distance measurements are also possible as mentioned in Jain [169] and Berkhin [170].

3 Coefficient training milestone is a pre-defined number of training iterations, i.e. one can define a coefficient training milestone as ten training iterations.

4 Energy error ($Ee$) is defined as a measurement of difference (or closeness or similarity) between the neural net estimated output ($y$) and the training data desired outputs ($t$) as described by

$$Ee = \frac{1}{2} \sum_{i=1}^{v} (t_i - y_i)^2$$

where $t$ and $y$ are vectors of same lengths, $v$.

5 Image component key or Image key or Image feature is an abstract part of an image determined in relation to the image that includes it. An image component is one of the individual parts of which an image is made up; especially a part that can be separated from or attached to an image. For facial image, image component keys are eyes, nose, and mouth, for example.
Image correlation indicates the strength and direction of a linear relationship between two random images of the same dimension. In general statistical usage, image correlation or image co-relation refers to the departure of two images from independence. Informally, it is a measure of how well an image matches a spatial-shifted version of another image, as a function of the amount of spatial shift.

\[
Correlation = \frac{(\sum_i a_i b_i)^2}{\sum_i a_i^2 \sum_i b_i^2}
\]

Image feature refers to parameters of an image object such as size, shapes, relative locations, textures, grey tones, colors, marks, etc.

Image texture is surfaces (the sequence of faces) that can contain data corresponding to not only a color but can be a virtual canvas for a picture or other image. Informally, a texture is an image that can be applied (“texture-mapped”) onto an object’s surface or particle.

Structural evolution decisions are responsible for neural net structural evolution involving four parameters (if applicable):

1. the number of filter inputs taps,
2. the number of filter feedback taps,
3. the number of hidden neurons, and
4. the number of hidden layers.

Each decision dictates the increment, decrement, or no-change modification for the corresponding structure parameter.

Target application or Intended application refers to an engineering problem which is formulated into a well-defined problem application. In this work, an intended application is a potential retargetable information processing problem that we can apply the cognitive processing on evolutionary platform to solve for possible solutions.
Training iteration is a process to train a neural net by presenting all training data (input-output pairs) to the neural net back-propagation algorithm one-by one.
Appendix B

Performance Metrics

The following metrics are proposed to measure the system performance. Each metric uses the following variables in its computation. During the system setup phase, we assume that we build \( n \) target profiles and \( n \) target signatures of length \( r \) from \( s \) sensors. Thus for the system architecture we need \( s \) feature extractors, \( r \) feature interfusors and \( n \) signature associators, Figure 3-6, Figure 3-7, Figure 3-8. The training set consists of system input-output data pairs. The inputs \((I_k)\) are the preprocessed sensory data from the \( s \) sensors of the corresponding target signature \((O_k)\) and its identification \((Q_k)\). Formally, the training set \((T)\) can be expressed as
\[
T = \{(I_k, O_k, Q_k) | I_k \in \mathbb{R}^{s+}, O_k \in \mathbb{R}^r, Q_k \in \mathbb{R}, k = 1, \ldots, n\}.
\]

**Cognitive Error.** The energy error of the cognitive recall (classification / identification), or the cognition error–\( EC \), measures the accuracy of the system in term of the energy level between the system cognitive recall (classification / identification) estimated output \((\hat{h} = \{h_1, \ldots, h_n\})\) and the actual target
identification \((Q_1, \ldots, Q_n)\) in the training set. We can express the identification error as 
\[ EC = \frac{1}{2} \sum_{k=1}^{n} (Q_k - h_k)^2. \]

**Fusion Error.** The energy error of the target (recognized) signature fusion (fusion error–\(EF\)) measures the accuracy of the system in term of the energy level between the system signature fusion (recognized / interfused) estimated output \((\hat{g} = \{g_1, \ldots, g_r\})\) and the actual target signature \((O)\) in the knowledge base. We can express the fusion error as 
\[ EF = \frac{1}{2} \sum_{k=1}^{r} (O_k - \hat{g})^2. \]

**Training time.** The training time of the system training measures the length of time the system needs to store all given target signatures in the neural net weights with a specific level of the errors (cognitive error and fusion error). The training time depends on two factors: the operation complexity and the computational unit capability (speed). The operation complexity will be discussed later.

**Target capacity.** The target capacity measures the maximum number of targets which the system can classify/identify with a specific level of errors (cognitive error and fusion error). From neural net literature [206, 16, 207, 208], the target capacity is linearly proportional to the size of the neural nets. Hence, we may assume that the system target capacity is dependent on all neural net sizes. However, we need to verify this claim with real experimentations for this system.

**Memory complexity.** It measures the amount of memory required for all neural net weights and structures during the system normal operation and system
training.

**Operation complexity.** It measures the amount of operation cycles required to compute one target classification/identification during the system normal operation and system training.

The cognitive processing system architecture is designed for modularity and portability. The system performance is directly dependent on the complexity aggregate of its components. The same performance metrics are applicable to the system components and the system as well.
Bibliography


[13] ACM, “Acm special interest group on artificial intelligence; acm special interest group on genetic and evolutionary computation.”
[14] IASTED, “Artificial intelligence and application; computational intelligence.”


[146] S. Ramanathan, S. K. Nandy, and V. Visvanathan, “Reconfigurable filter co-
processor architecture for dsp applications,” *Journal of VLSI Signal Process-


filter,” *IEEE Transactions on Circuits and Systems–II: Express BriefsIEEE 
Transactions on Circuits and Systems–II: Analog and Digital Signal Process-

dynamic partial reconfiguration,” in *International Symposium on Circuits 

[150] K. A. Vinger and J. Torresen, “Implementing evolution of fir-filters effi-
ciently in an fpga,” in *NASA/DoD Conference on Evolvable Hardware*. IEEE 

fir filter,” in *IEEE International Symposium on Circuits and Systems*, vol. 2. 

[152] D. Seidner, “Optimal fir filters with almost linear phase,” in *IEEE Interna-
tional Conference on Acoustics, Speech, and Signal Processing*, vol. 6. IEEE, 


