AUTONOMOUSLY RECONFIGURABLE ARTIFICIAL NEURAL NETWORK ON A CHIP

by

Zhanpeng Jin

M.S., Northwestern Polytechnical University, 2006

B.S., Northwestern Polytechnical University, 2003

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2010

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This dissertation was presented

by

Zhanpeng Jin

It was defended on

June 25, 2010

and approved by

AUTONOMOUSLY RECONFIGURABLE ARTIFICIAL NEURAL NETWORK ON A CHIP

Zhanpeng Jin, PhD

University of Pittsburgh, 2010

Artificial neural network (ANN), an established bio-inspired computing paradigm, has proved very effective in a variety of real-world problems and particularly useful for various emerging biomedical applications using specialized ANN hardware. Unfortunately, these ANN-based systems are increasingly vulnerable to both transient and permanent faults due to unrelenting advances in CMOS technology scaling, which sometimes can be catastrophic. The considerable resource and energy consumption and the lack of dynamic adaptability make conventional fault-tolerant techniques unsuitable for future portable medical solutions.

Inspired by the self-healing and self-recovery mechanisms of human nervous system, this research seeks to address reliability issues of ANN-based hardware by proposing an *Autonomously Reconfigurable Artificial Neural Network (ARANN)* architectural framework. Leveraging the homogeneous structural characteristics of neural networks, ARANN is capable of adapting its structures and operations, both algorithmically and microarchitecturally, to react to unexpected neuron failures. Specifically, we propose three key techniques — *Distributed ANN*, *Decoupled Virtual-to-Physical Neuron Mapping*, and *Dual-Layer Synchronization* — to achieve cost-effective structural adaptation and ensure accurate system recovery. Moreover, an ARANN-enabled self-optimizing workflow is presented to adaptively explore a "Pareto-optimal" neural network structure for a given application, on the fly.

Implemented and demonstrated on a Virtex-5 FPGA, ARANN can cover and adapt 93% chip area (neurons) with less than 1% chip overhead and $O(n)$ reconfiguration latency. A detailed performance analysis has been completed based on various recovery scenarios.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

This dissertation is intended to record and present one of my major research efforts during the Ph.D. study at the University of Pittsburgh. Fortunately, it has successfully finished after many months and years of hard work. However, it is the faculty, friends, and family members who have helped me to complete this dissertation. I would like to express my gratitude to these individuals for their support and assistance. Although I hope to list all people who offered their valuable help and suggestion to me and contributed to the extraordinary experiences I have been fortunate to enjoy over the past three years, I would inevitably leave out deserving colleagues, friends, and relatives and thus I am eager to show my sincerest respect and appreciation to all of "anonymous heroes" firstly.

The first person I would like to appreciate is my thesis advisor, Dr. Allen C. Cheng, for his constant guidance, encouragement, assistance, and support. Dr. Cheng not only put a lot of serious efforts into guiding my research but also spent tremendous time to proofread and correct all my works. Without him, the accomplishment of this dissertation would otherwise have remained only a dream. I have been fortunate enough to work with Dr. Cheng and have learned a lot from him. I've been deeply impressed by his admirable academic ambition, broad scientific view, remarkable enthusiasm, and exceptional intelligence full of new ideas. It is unimaginable how hard to mentor a foreign student and help him grow to be both a critical thinker and a qualified researcher. He spent countless nights in editing my manuscripts and confirming every aspect of research results to be tiniest level of detail. I can still remember the scenario when I received the revision of my first paper draft from Dr. Cheng. Every page is full of red markers with the modifications and comments from a single word to the whole section. Thanks to the extensive training offered by Dr. Cheng, I was excited to see less and less "red markers" shown in my paper drafts. More importantly, he constantly gave me valuable suggestions and kept me abreast with the state of the art electronic and computer techniques which greatly benefit my research. Furthermore, he has also taught me many important lessons in life which will always guide me throughout my career.

Thanks to Dr. Steven P. Levitan, Dr. Zhi-Hong Mao, Dr. Marlin H. Mickle, Dr. Shi-Kuo Chang, and Dr. Wenyan Jia for serving on my doctoral dissertation committee.

Dr. Levitan is one of the most engaging, humorous, and friendly professors I've ever met. His incredible sense of humor and jokes made the taxing classes much more joyful and refreshing. I really enjoy the courses with him and appreciate his extensive expertise in VLSI and EDA areas. On the other side, Dr. Levitan is also among the most critical and strict faculty members in the pursuit of scientific excellence. The challenging questions he raised and inspirable comments were invaluable help to me while working on my dissertation.

Holding the joint Ph.D. degree from two big-name schools — MIT and Harvard, Dr. Mao is well known for his solid background in mathematics, control systems, computation intelligence, and neuroengineering. Dr. Mao is very approachable and warms to students readily. He has gained great reputation from students for his excellent teaching. He has guided me as a friend through his wisdom and valuable experience. I am really thankful for the generous time and promptness he offered in giving me suggestions and solutions to overcome research obstacles. I also thank him for being open and sharing with me his wonderful experience and perspectives on the academic careers.

As one of the most prestigious faculty members in our department, I am sincerely impressed by Dr. Mickle's unremitting devotion to research and teaching. Over the past half century, he has observed and actively engaged into the incomparable development of electronic and computer techniques. Now, he continues to share his wealth of experiences to all his students. In his computer architecture class, Dr. Mickle always has the wisdom and experience to simplify situations and things that seem so complicated to me.

Thanks to Dr. Chang and Dr. Jia, the external committee members, for their providing me significant comments and suggestions from other perspectives and helped me sharp my thinking. I could not work out this thesis without their dedicated help, guidance and support.

I would like to thank the Department of Electrical and Computer Engineering for the tremendous graduate education I have received in the past three years and especially those administrative staff for their kind help and dedicated support. Several individuals deserve special mention for their long valuable help during my studies in ECE. I thank William McGahey and Jim Lyle for their long, timely technical assistance with lab infrastructure. I owe a special debt to Theresa Costanzo and Sandy Weisberg for their kind understanding and patience to help me solve various problems and personal matters.

I've been especially fortunate to know and work with Yuwen Sun, Shimeng Huang, Joseph Oresko, Heather Duschl, Kingsley Adeoye, Jun Cheng, and Timothy Sestrich in the ACT Lab. I really enjoy the wonderful time with all these good friends, either in research or in pleasure. They are the constant source of my inspiration. Their intelligent ideas and technical skills have inspired and boosted lots of my work.

I would like to gratefully and sincerely thank Michael and Bronwynn McAdams, Bill and Phyllis Sutton (my beloved grandpa and grandma in America), Carol McAdams, and all members in Sutton's family and McAdams's family. It is your strong faith in the faithful God, admirable love, friendly help, and selfless support that help me survive in another country and feel like living at another home with many cordial family members. I am so grateful for your accompany and all you did in the past three years.

My family has been a tremendous source of love, encouragement, and inspiration. I thank my parents, Yuejie Jin and Suxia Guo, for their faithful love and persistent supports allowing me to continue my earnest pursuit in sciences. Their kind indulgence has kept me working towards my ambitious goals through my entire life. Also, I wish to express my earnest gratitude to my mother in law, Ms. Hongxu Zhao. She provided me with unending understanding, encouragement and immeasurable support on my studies, and endured the suffering of missing her daughter alone.

Finally, and most importantly, I would like to express my deepest appreciation to my wife Jie Yin. Your unwavering love, great support, helpful encouragement, quiet patience, kind indulgence, and significant sacrifice were undoubtedly the bedrock upon which I can continue my academic pursuit in Pittsburgh in the past three years. I am very thankful for your faithfully staying besides me during the most difficult and confusing time of my life. Without you, I can't believe this work could be finished as fast and well. I love you so much!

1.0 INTRODUCTION

1.1 RESEARCH MOTIVATION AND PROBLEM STATEMENT

Throughout the history of digital electronics, the technology has improved exponentially over time. The performance of devices is roughly doubling every 18 months because transistor size and cost of chips have shrunk at an impressive pace. Unrelenting advances in the transistor density of integrated circuits have resulted in a large number of engineered systems with diversified functional characteristics to meet various demands of the human life, ranging from micro-embedded devices, implantable medical devices, smart sensors, to critical infrastructure. Correspondingly, the complexity of system-level design for these increasingly evolved engineered systems is further compounded when interdisciplinary requirements are included, for example, massive integration and interconnection between components and subsystems, feedback and redundancy. The increasingly shrinking electronic technology and the compound complexity in these systems have resulted in substantial increases in both the number of hard errors [78], mainly due to variation, material defects, and physical failure, as well as the number of soft errors [259], primarily due to alpha particles from normal radiation decay, from cosmic rays striking the chip, or simply from random noise. Although these complex systems are designed to guarantee robust operation to the events that have been anticipated and accounted for in the design blueprint, unfortunately, most engineered systems still operate under great risk of uncertainty. To ensure the appropriate operation of complex systems under highly unreliable circumstances, a new paradigm for design, analysis and synthesis of engineered systems is needed. It is therefore imperative that system designers build robust fault-tolerance into computational circuits, and that these designs have the ability to detect and recover the damages causing the system to process improperly or even disabled.

Recently, the concept of *autonomous reconfigurability* has emerged and evolved to be a promising mechanism for ensuring appropriate operational levels during and after unexpected natural or man-made events that could impact critical engineered systems in unforeseen ways or to take advantage of unexpected opportunities. *Autonomous reconfigurability* refers to a system's ability to change its structure and operations or both in response to an unforeseen event in order to meet its objectives. This concept can be realized and advanced using the powerful state-of-the-art computational platforms and techniques, including a collection of hardware devices, software, networks, and ubiquitous computation, which can provide the capability for embedding reconfigurability into complex engineered systems. Thus, as one fundamental infrastructure, a flexible hardware substrate is required to support the idea of "reconfigurability", providing considerable space and performance for very large-scale integration of heterogeneous, multi-functional circuitries and enough flexibility for allowing the adaptation mechanism to modify the system. Specifically, the hardware substrate must provide a mechanism to accommodate redundant system components or design elements, to evaluate and change different possible topologies/connections dynamically, to be easily accessed internally and externally at any point in time, and to allow a sufficiently wide search space.

Artificial neural network (ANN), an established bio-inspired computing paradigm, mimics its biological counterpart in the human brain to provide self-adaptive flexibility and powerful learning ability. Such neural networks usually involve a highly structured network of simple processing elements (neurons), which can exhibit complex global behaviors, determined by the synaptic connections between processing elements and specific element parameters. The true power of neural networks lies in their abilities to represent both linear and non-linear relationships and to learn these relationships directly from the data being modeled. Given its considerable capability in recognizing complex patterns, ANN has proved very effective in a variety of real-world problems [230] and has been particularly investigated for emerging biomedical applications [15, 65, 121, 170]. For instance, many successful applications of neural networks on biomedical problems have been extensively reported and demonstrated, including electrocardiography (ECG) [47, 120, 119, 134, 135, 208, 261], electromyography (EMG) [29, 33, 88, 106, 210, 292], electroencephalography (EEG) [56, 140, 171, 175, 265, 267], medical speech [132, 243, 309] and image processing [149, 167, 182, 234], etc. Accordingly, many dedicated ANN-based devices and systems have been developed using off-the-shelf hardware to facilitate the healthcare and clinical treatment. As people become more active in monitoring their own health conditions and the remarkable development of telemedicine and pervasive healthcare techniques in recent years, ultra-portable and ultra-reliable ANN-based medical systems have become of great interest to the whole society.

Specifically, for the portable goal, it may not be optimal to design a highly redundant system providing exhaustive coverage on any system failure. Emerging smart biomedical devices will be either worn by people or implanted into human body, thus they are expected to play significant roles in non-stop routine monitoring and directing some medical response mechanisms, such as implanted pacemakers and defibrillator [28, 155, 235, 236, 296, 297], wearable functional electrical stimulation (FES) devices [76, 77, 282, 283, 299], or prosthetic limbs [9, 192, 160, 263]. It is manifest that how severe problems can be caused, sometimes people would die from this, if the devices can not work properly due to unexpected faults. Meanwhile, it is also unacceptable that the systems have been out of order for a long time and the patients have to wait for the repair or replacement of the broken devices, not to mention that sometimes it would be really difficult to replace a device without invasive surgery.

Like other electronic systems, ANN-based systems are also increasingly vulnerable to both transient and permanent faults [157] which sometimes can be catastrophic, especially for medical applications. Conventional fault-tolerant techniques applicable to ANN-based systems include spatial redundancy, temporal redundancy, and coding [227]. Those techniques, such as Error Correction Coding (ECC) [41, 53, 83], Dual Modular Redundancy (DMR) [82, 278] or Triple Modular Redundancy (TMR) [215, 244, 257, 260], usually consume considerable system resources and energy, which can be prohibitive to meet the strict requirements of next-generation portable or implantable medical applications. Moreover, their lack of dynamic adaptability makes their protection effective only against faults that can be conceived at the design stage.

Inspired by the precise, systematic, sophisticated, supremely intelligent, and essentially AR-based automatic recovery mechanisms of the mammalian Central Nervous System (CNS) to react to unexpected injuries or diseases, we would like to explore the possibility to mimic

CNS's faulty reaction strategies and to develop a cyber-enabled artificial neural network environment with self-healing and self-optimizing capabilities. This dissertation proposes a novel *Autonomously Reconfigurable Artificial Neural Network (ARANN)* architectural framework. Leveraging the inherently homogeneous structural characteristics of neural networks, ARANN is capable of adapting ANN's structures and behaviors, both algorithmically and microarchitecturally, to react to unexpected faults at any neuron.

In this study, we will examine and investigate various techniques to help build a novel flexible and reliable artificial neural network platform, which is particularly suitable for next-generation mission-critical applications requiring high degree of reliability and portability.

1.2 SCOPE OF THESIS

1.2.1 Research Hypothesis and Aims

It is well demonstrated that the reliability issues of future highly integrated electronic systems have become increasingly severe. While there are steadily increasing demands on the reliable, fault-tolerant devices to improve the quality of people's lives, particularly in the biomedical domain to facilitate the medical treatment and prevention of individuals. In this study, we will focus on a particular important computing technique widely used in biomedical applications — artificial neural network — and investigate the possibility of building reliable and fault-tolerant ANN-based systems from a biologically inspired perspective. Orthogonal with conventional electronic reliability design techniques, we would like to explore an alternative way to augment the fault-tolerance and resilience of ANN-based hardware systems, leveraging the inherently homogeneous structural characteristics of neural networks. The research hypothesis of this study is that:

"*Reliable and fault-tolerant ANN-based next-generation biomedical platforms can be realized by incorporating appropriate bio-inspired autonomous reconfigurability.*"

To test this hypothesis and demonstrate the feasibility of maintaining appropriate levels of operation and performance of ANN-based systems by autonomously reconfiguring its structures and adapting its operations, in case of expected faults occurring at any computation nodes (neurons), we have several specific research aims shown as follows.

1. To enable ANN-based system to make appropriate structural adaptations in response to unexpected failures of computational neuron nodes.
2. To make ANN-based system determine a well balanced (Pareto-optimal) structure in an online and autonomous manner, when functioning accuracy, generalization capability and power consumption are all of great concern to designers and users.
3. To reduce the latency and overhead of structural adaptation and realize cost-effective system reconfiguration.
4. To demonstrate the efficacy and efficiency of the proposed autonomously reconfigurable artificial neural network architectural framework on a real hardware platform, aiming at a targeted application.

1.2.2 Addressing ANN's Recovery Issue

Inspired by the precise, systematic, sophisticated and supremely intelligent automatic recovery mechanism of the mammalian Central Nervous System (CNS) to react to unexpected injuries or diseases, we would like to explore the possibility to mimic CNS's faulty reaction strategies and to develop a cyber-enabled artificial neural network environment with self-healing capabilities. Artificial neural network is essentially a highly scalable and parameter non-sensitive architecture, which means the overall system performance is determined by a large set of homogeneous neuron units and their associated synaptic connections, thus the change of a specific synaptic connection or the adding/removing a specific neuron unit will not cause tremendous effects on the whole ANN system. The inherent characteristics of neural networks make the principles of Autonomously Reconfigurability perfectly applicable to the ANN systems and help ANN systems meet the extremely stringent requirements on reliable operations. Given the fact that most previous reliability-aware studies usually enhance the fault tolerance capability of state-of-the-art hardware systems based on either space- or time-redundancy techniques, in this study, we hope to address the reliability issues of emerging ANN-based hardware from another perspective.

In principle, the ultimate goal is to achieve a reliable solution and at least maintain appropriate operational capabilities by making ANN system capable of adapting its structure or operations in response to an unforeseen event. This strategic target essentially involves an optimal trade-off among system performance, reliability requirements, and associated costs. Instead of preparing a lot of identical backup hardware components to fill in the vacant positions in case some components are physically damaged, our proposed system would be capable of dynamically determining an optimal structure and connections of each individual neuron unit for the ANN system, as well as adaptively finding and incorporating available neuron resources to maintain the best achievable performance of the affected ANN system. Specifically, in a similar way as CNS's recovery process in case of a acquired brain injury, the proposed self-healing ARANN architecture can immediately adapt the system structure to disconnect the damaged neuron unit from the main network, if any error has been reported by the fault detector, and then involve new neuron units into the network to maintain the desired performance if any available neuron units are found. Otherwise, if no further neuron resources are available, the ARANN will continue its normal operation in a compromised mode caused by the slightly fewer neuron nodes contained by the current ANN system. One of the most profound benefits of the proposed ARANN is the opportunity to help ANN system react to any unexpected harmful events in an autonomous, on-line, and efficient manner without halting system execution and introducing considerable redundancy.

To better utilize the massively parallel processing nature of neural networks and facilitate their structural adaptation, we propose a *Distributed Artificial Neural Network (DANN)* architecture. Unlike prevailing Centralized ANN implementation, which usually features a "master-slave" system consisting of a highly-centralized, heavy-weight controller and a group of rather simple computational nodes (neurons), DANN is mainly made up of a lightweight topological & algorithmic controller and a mass of highly independent, autonomic, smart neuron units. Alleviating the computational burden from one central controller to a number of homogeneous neuron nodes, DANN can significantly improve the system performance by maximizing the degree of neuron-level parallelism. Also, DANN greatly reduces the data and control dependency between the central controller and all neurons, which provides a more flexible architectural infrastructure for the ANN structural adaptation.

Given the loosely coupled computations and communications enabled by DANN, we present a novel *Decoupled Virtual-to-Physical (V2P) Neuron Mapping* strategy to implement a cost-efficient system adaptation scheme. Specifically, we propose a "neuron virtualization" by abstracting away the direct connections between ANN controller and all physical neuron units, and inserting a flexible V2P neuron mapping block to determine appropriate connections between virtual and physical neuron ports, according to the desired number of virtual neurons specified by the algorithm and the availability of individual physical neurons implemented on the hardware. With such a decoupling scheme, the real spatio-temporal connections of "physical neurons" is transparent to the controller that handles "virtual neurons". A detected faulty neuron can be timely removed from the neural network by changing the corresponding V2P mapping scheme to swap the faulty neuron with a spare neuron. The proposed Decoupled V2P Neuron Mapping has successfully addressed the reconfigurability and adaptability issues of conventional neural network implementations. It provides a convenient way to achieve the resource-efficient neuron reuse. More importantly, it indicates the possibility of increasing ANN's reliability by automatically reconfiguring and revising its structure in case one or more physical neurons are damaged.

Furthermore, the standard ANN learning process involves a large amount of training epochs, training patterns, and training stages (i.e., feed-forward calculation, back-propagation, and weight updating) and is also highly data dependent, since the magnitudes of synaptic weight changes highly reply on the calculated performance errors, which is iteratively determined by the synaptic weights updated in the training procedure of either previous input pattern or most recent epoch. Considering the severe influence of a faulty neuron on the whole neural network due to the inevitable time delays among the occurrence, detection, notification, and treatment of faulty neurons, a more accurate system recovery scheme besides the systematic reconfiguration is highly demanded to guarantee both the successful recovery of ANN systems in both physical structures and functioning accuracy. In the ARANN architectural framework, we ensure a smooth, accurate and consistent recovery, no matter when an unexpected fault is detected, by proposing the *Dual-Layer Memory Synchronization* mechanism, which includes a fine-grained memory and a coarse-grained memory maintaining and synchronizing relevant ANN state information on a stage or epoch basis respectively.

1.2.3 Addressing ANN's Optimization Issue

Artificial Neural Networks (ANNs), since its earliest emergence about half a century ago, have been extensively studied and broadly used in a wide variety of applications, such as biomedicine [65, 170], industrial control [172, 201, 269], finance [139, 255, 307], engineering [43, 142], and computer science [89, 252]. Along with the remarkable efforts researchers have made to discover more effective ANN algorithms for some as of yet unsolved problems, another important research question of great concern is how to find and determine the best structure and configuration for a given ANN algorithm. Usually, for a standard fully-connected multi-layer perceptron neural network, the most critical parameters include the number of hidden layers and the number of neurons in each layer. It has been widely investigated and demonstrated that, with any of a wide variety of continuous nonlinear activation functions, one hidden layer with an arbitrarily large number of neurons suffices for the "universal approximation" property discussed by Hornik [112, 113, 114] and Bishop [19] respectively. In this case, the number of neurons in the only hidden layer becomes the only significant parameter that determines MLP's behavior and performance. Unfortunately, there has not been any theory yet to precisely determine the right (optimal) number of hidden neurons used by MLP for a specific problem. Although researchers have proposed many criteria or algorithms to help ANN users explore an optimal structure, such as the Akaike's Information Criterion (AIC) [5], Network Information Criterion [195], and the exploration of best number of hidden neurons [80, 173, 291], it is still in early stage to widely apply all these algorithms onto real problems due to either their extremely complex algorithmic computations or application-dependent characteristics. Until now, most of previous studies using neural networks have still highly relied on the science of experience or extensive experimental trials. Therefore, a practical issue of using ANNs is how to determine a optimal ANN structure, particularly the number of hidden neurons in the network. In general, the neural network may not learn the presented problem well if it is too small. On the other side, an over-sized network may lead to over-fitting and poor generalization performance [98]. Thus, as we presented before, it is highly desired that the ANN systems can find appropriate network architecture automatically under the guidance of certain algorithms.

To solve real-world problems using ANNs, it usually requires the use of highly structured networks of a rather large size. A rule of thumb for obtaining good generalization capability is to use the smallest system that will fit the data [233]. Because a neural network with minimum size is less likely to learn the idiosyncrasies or noise in the training data, and may thus generalize better to new data [103]. Since there has not been any theory capable of directly determining the best size of neural networks, we should search and find an optimal network structure by comparing various potential candidates according to a certain evaluation criterion. One effective and efficient approach is so-called *network pruning*. It starts with a rather large MLP with sufficient neuron units for the given application, and train the initial system using a common learning algorithm until an acceptable training accuracy achieved. After that, some inactive neurons will be gradually removed or certain synaptic weights will be eliminated in a selective and orderly fashion. This key idea is to iteratively evaluate the trade-off between the training accuracy and the structural complexity of ANN systems and then select the optimal structure providing reasonable accuracy with the least design complexity.

Although such type of optimization strategies has been extensively investigated and used in software implementations of neural networks, there has not been any neural hardware capable of dynamically optimizing its structure and providing efficient solutions for different applications, because most neural hardware were developed for certain applications only and they are reluctant to evolve into a more efficient shape. However, for emerging wearable biomedical devices and future pervasive healthcare, a highly integrated, multi-functional, ultra low-power, ultra-portable, extraordinary reliable hardware platform is mandatory. As one of the most important and promising techniques, ANN-based hardware is also expected to fit different applications in a more power-efficient manner. One possible solution to achieve this goal is to make ANN adaptable and reconfigurable and thus determine the system structure according to specific requirements and design trade-offs between performance measure and complexity overhead. Leveraging the reconfigurable and adaptable architectural infrastructure provided by ARANN, we incorporated the concept of neural network pruning into ARANN and proposed a *Self-Optimizing Artificial Neural Network (SOANN)*, making use of ARANN's incomparable capabilities of connecting and disconnecting any physical neuron unit to/from the main network on the fly. Instead of determining an "optimal" neural network structure for one certain application by the off-line analysis, the ARANN architecture will be able to evaluate the system cost involving both performance measure and complexity overhead, and then adaptively explore the most optimal network structure with the appropriate performance tradeoff. In summary, the proposed ARANN-based self-optimization approach is capable of helping users further shape the structure of neural networks and remove unnecessary (or "redundant") neurons which have little or no influence on the overall network performance.

Another major motivation to develop a flexible neural network platform with the capability of adapting and optimizing its structure in an autonomous manner is the increasing demands on the more diversified neural network systems. It is well known that ANNs have had very broad applications in biomedical domain. Most of previous studies usually involve extensively off-line analysis of experimental data and then propose a "supposedly" optimal neural network model for that particular problem. This may be the case for a very concentrated study with relatively few variables or parameters involved. However, as the rapid development of biomedical sciences, more sophisticated clinical techniques have been invented to provide more accurate diagnostic solutions and address certain highly intricate medical conditions. It is well known that the human body is one of the most complex elements in the universe and thus any medical condition can not be purely caused by one or a few clinical parameters. Therefore, such type of emerging comprehensive diagnostic systems usually involves the exhaustive analysis on a variety of biomedical parameters. For instance, Hudson and Cohen [49, 122] proposed a hybrid system in which biomedical signal data (e.g., ECG, EEG, and other clinical parameters) can be incorporated for developing higher-order medical decision systems and demonstrated increased sensitivity, specificity, and accuracy. Usakli *et al.* [287] also presented the possibility of involving both electroencephalogram (EEG) and electrooculogram (EOG) for the development of future Human-Computer Interface (HCI) or Brain-Computer Interface (BCI). In these cases, although a fixed neural network structure may be able to provide reasonable accuracy for modeling one certain bio-signal as studied in many previous work, it is highly desired that a reconfigurable neural network platform can be developed and adapted to meet different needs and characteristics of various biomedical parameters in an on-line manner. For example, one research scenario is to develop a hybrid healthcare assistive system, where ANN-based platform can be dynamically adapted and autonomously optimized for different usages, such as using parameter A to detect condition I, using parameter B to detect condition II, or using parameters A and B to monitor condition III, etc. In addition, a self-optimizing neural network can provide more flexibility to find an optimal model for a certain problem. The exploration of an optimal biomedical model is extremely challenging and sometimes even unfeasible, because many biomedical parameters are closely correlated and interacted, and thus it is really difficult to accurately identify the set of "influential" parameters. One possible solution is to tentatively investigate potential involved variables and adaptively find the most optimal model based on the identified variables. For example, Cecotti and Gräser [31] proposed to use neural network pruning strategy to reduce the number of electrodes and to select the best electrodes in relation to the subject particularities for a P300 Brain-Computer Interface application. It is shown that, even the involved biomedical parameters can not be accurately determined without adaptive evaluations, not to mention the optimal structure of neural networks used to model their behaviors. A reconfigurable neural network platform with the on-line self-optimizing capability will provide promising performance benefits for next-generation sophisticated biomedical solutions and significantly reduce the off-line optimization analysis efforts.

As electronic circuits' speeds and circuit densities continuously increase, circuit board power density increases as well and thermal management becomes an increasingly significant part of system design [25]. During the development of a large-scale circuit board, thus the thermal design aspects have proved crucial to its reliable operation. Reducing thermally induced stress and preventing local overheating remain major concerns when optimizing the capabilities of modern system chips [24]. However, such thermal-efficient approaches will usually bring considerable loss of performance, which is also critical to the increasingly computation-intensive applications. Therefore, seeking an effective way to balance the requirements on the high computational performance and the reliable operations with efficient power management has been of great interest to the academia. The employment of reconfigurability design concept can bring extra benefits in further addressing the reliability issues during the system execution. Given the flexible adaptability provided by ARANN, we propose to further augment system's reliable operation and prevent the system from overheating without loss of performance by providing more modular design options that can be conveniently loaded and swapped into the main system. These design options may offer different performance/power tradeoffs and many other controllable diversified characteristics. With the support of the proposed Virtual-to-Physical Neuron Mapping, such swaps between characteristics-specific modules can be accomplished within a little while. Once the system has been cooled down by switching to power-efficient design modules, the high-performance system components can be now reloaded into the system again. In this way, the complex system can achieve an optimal balanced tradeoff between the intensive performance demands and the robust reliability requirements.

1.2.4 Addressing ANN's Adaptation Cost Issue

As we mentioned before, given the desired number of neuron units (determined by the ANN Controller) and the locations of potentially damaged neurons (designated by the Error Detector), the Virtual-to-Physical (V2P) Neuron Mapper will establish connections between the virtual neuron ports and corresponding physical neuron units. There are generally two cases associated with such V2P mapping process. The first case is that the available (physical) neuron units in hardware are more than the desired (virtual) neurons specified by the ANN Controller, thus like those faulty neurons, some neuron units will not be enabled and used in the current ANN structure. The other case is that the available physical neuron units are not enough to meet the needs of the ANN Controller any more, probably due to gradually increased damage on hardware. In this case, the V2P Mapper will exhaustively search those still "healthy" neuron units and involve all of them in the current ANN structure. Also, the V2P Mapper will return the number of currently involved physical neuron units and a feedback signal back to the ANN Controller to tell users that the system is now running in a "Compromised" mode and the level of damage on hardware.

It is shown that the V2P Neuron Mapper is one of the most critical components within this Autonomously Reconfigurable Artificial Neural Network (ARANN) architecture and also the major element which introduces extra time and space overhead to the ANN system.

Considering the possibility that the electronic reliability issues will become increasingly severe and the exponentially growing needs of more versatile, easily configured ANN hardware, it is highly desired to design and implement a fast, flexible, accurate, and resource-efficient V2P mapping block which can be integrated into our ARANN architecture. In this thesis, we explore several different V2P mapping design solutions from various perspectives and then analyze their specific characteristics (i.e., performance, implementation efficiency, and potential overhead) and applicabilities to pursue the lowest time and space overhead associated with the demonstrated autonomous reconfiguration capability. We propose four V2P design strategies: 1) Adaptive Physical Neuron Allocation ("V2P Mapper"), 2) Cache-Accelerated Adaptive Physical Neuron Allocation ("V2P Mapper w/ Cache"), 3) Virtual-to-Physical Neuron Mapping Memory ("V2P Memory"), and 4) Mask-Based Virtual-to-Physical Neuron Mapping Memory ("Mask-based V2P Memory").

According to thorough comparison of results, it is clearly shown that there isn't a perfect design choice and all these four design strategies have distinct characteristics in design complexity, resource requirement, time overhead, and applicability to various scales of problems and thus have their own advantages and limitations. Since in this thesis we only investigated and implemented a small-scale ANN system for a relatively simple biomedical application (see section 3.3) and we also assume a relatively low defective probability for our ANN system, thus the *Cache-Accelerated V2P Mapper* seems to be a good design choice and has been used in all of our experiments thereafter.

1.3 CONTRIBUTIONS

Computer systems may fail in any number of ways, thus some certain levels of fault-tolerance are extremely necessary, and particularly critical for emerging biomedical portable/implantable systems due to their difficult system rebuilding and physically invasive procedures. Artificial neural network (ANN) has proved to be effective in a variety of biomedical applications and many ANN-based medical solutions today have been demonstrated using off-the-shelf hardware. Due to unrelenting advances in technology scaling and large scale integration, these systems are increasingly vulnerable to both transient and permanent faults which sometimes can be catastrophic, especially for medical applications. Conventional fault-tolerant techniques applicable to ANN-based systems usually consume considerable system resources and energy, which can be prohibitive to meet the strict requirements of next-generation ultra-portable or implantable medical applications. Moreover, their lack of dynamic adaptability makes their protection effective only against faults that can be conceived at the design stage.

Inspired by the precise, systematic, sophisticated, supremely intelligent, and essentially autonomous reconfiguration-based automatic recovery mechanisms of the mammalian Central Nervous System (CNS) to react to unexpected injuries or diseases, we would like to explore the possibility to mimic CNS's faulty reaction strategies and to develop a cyber-enabled artificial neural network environment with self-healing and self-optimizing capabilities. This dissertation proposes a novel *Autonomously Reconfigurable Artificial Neural Network (ARANN)* architectural framework. Leveraging the inherently homogeneous structural characteristics of neural networks, ARANN is capable of adapting ANN's structures and behaviors, both algorithmically and microarchitecturally, to react to unexpected faults at any neuron. More specifically, in a similar way as CNS's recovery process in case of a acquired brain injury, the proposed self-healing ARANN architecture can immediately adapt the system structure to disconnect the damaged neuron unit from the main network, if any error has been reported by the fault detector, and then involve new neuron units into the network to maintain the desired performance if any available neuron units are found. Otherwise, if no further neuron resources are available, the ARANN will continue its normal operation in a compromised mode caused by the slightly fewer neuron nodes contained by the current ANN system. Given the incomparable capabilities of connecting and disconnecting any physical neuron unit to/from the main network on the fly, ARANN will be able to evaluate the system cost involving both performance measure and complexity overhead, and then adaptively explore the most optimal network structure with appropriate design tradeoff.

The contributions of this dissertation research are threefold:

- First, we propose a novel biologically-inspired *Autonomously Reconfigurable Artificial Neural Network (ARANN)* architectural framework, capable of adapting ANN's structure and operations, both algorithmically and microarchitecturally, to react to unexpected faults occurring at any neuron. We demonstrate the effective and efficient self-healing and self-optimizing system adaptation methodologies on the ARANN, leveraging several architectural innovations which include the *Distributed ANN* architecture, the neuron virtualization technique with a *Decoupled Virtual-to-Physical Neuron Mapping*, and a *Dual-Layer Memory Synchronization* mechanism to ensure a smooth, accurate and consistent recovery of the highly structured neural network systems.

- Secondly, to further reduce the added time latency and resource overhead associated with ARANN's dynamic structural reconfiguration, we present and investigate four possible design solutions for the most critical component in the ARANN — *Virtual-to-Physical Neuron Mapping*. A thorough analysis and comparison have been performed on all of them to explicitly demonstrate their specific applicabilities.

- Thirdly, we verify the proposed ARANN using a real biomedical case study that presents an ANN-based model for limb endpoint locomotion prediction. And we also prototype the proposed ARANN on the state-of-the-art FPGA platform. Because of the demonstrated scalability and properties, the proposed ARANN architectural framework will be scalable to different scales of neural networks and can be deployed on either a single integrated circuit chip or a multiple processing elements environment, such as multi-FPGAs, multicore chips or chip-multiprocessors (CMPs).

In summary, the proposed ARANN architectural framework provides designers (particularly future biomedical system designers) with a new genre of highly integrated, multi-functional, ultra low-power, ultra-portable, extraordinary reliable neural network platform that can achieve self-healing and self-optimization through autonomous structural reconfiguration. The ARANN system either can be completely recovered or can be adapted into a "compromised" mode with a certain degree of performance tradeoff. Both two solutions can be achieved timely and will not stop the system execution at all. Actually, the most profound benefit of the proposed ARANN is the opportunity to help ANN system react to any unexpected harmful events in an autonomous, on-line, and efficient manner without halting system execution and introducing considerable redundancy.

1.4 THESIS OUTLINES AND ORGANIZATIONS

The reminder of this thesis is organized as follows:

Chapter 2 gives an overview of artificial neural network techniques, reviews all previous efforts on the hardware implementation of artificial neural networks, and discusses prior reconfigurable neural network design practices that are closely related to our study.

Chapter 3 provides a methodological overview on artificial neural networks and particularly examines the Multilayer Perceptron (MLP) including its architectural properties, interconnections, and back-propagation training algorithms. We also present a case study on the effective use of ANN in emerging biomedical applications — an ANN-based model for limb end-point locomotion predictions.

Chapter 4 presents the proposed Autonomously Reconfigurable Artificial Neural Network methodology framework. We describe the major architectural innovations and hardware infrastructure that support the ARANN, and demonstrate the effective and efficient self-healing and self-optimizing system adaptation methodologies on the ARANN.

Chapter 5 discusses the design issues and challenges when implementing a Multilayer Perceptron (MLP) neural network on the FPGA, given the relatively limited resources available. Specifically, we discuss the issues with regard to arithmetic representations, multi-purpose smart neurons, activation function implementation strategies, as well as the hierarchically bidirectional neuron/synapse-reused ANN implementations. The realization details are all demonstrated in this chapter.

Chapter 6 shows the experimental results and gives detailed analysis on the performance benefits and overheads of the proposed ARANN architectural framework. For comparison purpose, a case study on the aforementioned biomedical application of ANNs is also demonstrated; the performance results based on the proposed reconfigurable platform are compared with the data generated from MATLAB simulation.

Chapter 7 offers conclusions and future directions of this thesis research. This thesis involves several topics, and consequently, the work in this thesis could be continued and extended in a variety of directions, such as utilizing the platform-level reconfigurability or deploying ARANN onto multi-chip environment for more complicated applications.

2.0 RELATED WORK

This chapter describes prior and concurrent research related to this dissertation study. These relevant studies are presented under different categories based on the nature of the work. Along with detailed description and comparative analysis, what is also provided are insights explaining what the proposed ARANN is different from all prior work and why it is advantageous over those work.

2.1 HARDWARE IMPLEMENTATIONS OF ARTIFICIAL NEURAL NETWORKS

General-purpose computers are traditionally based on the von-Neumann architecture, which is essentially sequential. Artificial neural networks, on the other hand, significantly benefit from their massively parallel processing nature. In the past several decades, the performance of conventional von-Neumann processors has continued to increase dramatically and the up-to-date computing systems have been able to meet various increasing computational requirements. Thus, when the extraordinary processing performance is not particularly desired, most researchers or designers who widely explore ANN-based solutions to solve real problems, usually rely on the software implementation on a PC or workstation without any special hardware components or devices. A tremendous amount of work has been done in developing simulation environments for artificial neural networks on sequential machines [81, 196]. However, the software simulation cannot provide real-time learning and response when the emulated ANNs contain a large number of neurons and synapses, even on the fastest sequential machines. The inherently parallel nature of ANNs demands a more parallelized computational architecture capable of processing synaptically connected neurons simultaneously using multiple simple processing elements (PEs).

The idea of building the neural network on hardware platform is definitely not new and can date back to more than one decade ago. Some emerging specialized applications have motivated the use of application-specific neural network hardware. For example, a variety of low-cost consumer devices dedicated for certain applications (e.g., speech recognition) and analog neuromorphic devices (e.g., silicon retinas) have been introduced and developed to meet people's various demands on ANN-based solutions [163]. The development of digital neuro-hardware is driven by the desire to speed-up the simulation of ANNs and to achieve a better performance-to-cost ratio than general-purpose systems [251]. It has been demonstrated that the hardware implementations of artificial neural networks are able to take full advantage of their inherent parallelism and thus can achieve much better performance by orders of magnitude compared to their counterparts simulated in software.

In general, neural network hardware designers have followed two distinct approaches. One is to build a general, but probably expensive, system that can be re-programmed for many kinds of tasks, such as Adaptive Solutions' CNAPS (Connected Network of Adaptive Processors) [97], Siemens' SYNAPSE (Synthesis of Neural Algorithms on a Parallel Systolic Engine) [232], as well as the NESPINN (Neurocomputer for Spiking Neural Networks) [130]. Another approach is to build low-cost, application-specific chips that can handle computationally intensive and regular tasks effectively and efficiently for certain applications, such as IBM's ZISC [169]. Several overviews on available neural network hardwares and systems have been presented and published by Dias et al. [60], Moerland and Fiesler [188], and Lindsey [168], as illustrated in Table 1.

Many researchers initiated efforts to integrate neural networks of large sizes on a single chip [85, 231, 288], which can execute complex operations of neural networks at a higher speed and a lower per-unit cost compared to software implementations. As the dramatic development of integrated circuit technology, the main implementations of neural networks have been evolving and expanding from the original LSI circuits to the latest SoCs, 3D chips, FPGAs and digital/analog/mixed-signal VLSIs. In what follows, I will give a brief overview on the prior efforts on ANN hardware implementations and neural computing systems.

Table 1: Examples of Neural Network Hardwares

System	Architecture	Learn	Precision	Neurons	Synapses	Speed
			Analog Implementations			
Intel ETANN	FF ML[a]	No	6b×6b	64	10280	2 GCPS
Synaptics Silicon Retina	Neuromorphic	No	N/A	48×48	Resistive net	N/A
			Digital Implementations			
Philips Lneuro-1	FF ML	No	1-16b	16 PE	64	26 MCPS[b]
Hitachi WSI	Wafer, SIMD	BP[a]	9b×8b	144	N/A	300 MCUPS[c]
Siemens MA-16	Matrix ops	No	16b	16 PE	16×16	400 MCPS
IBM ZISC036	RBF	ROI[a]	8b	36	64×36	250 kpat/s
SAND/1	FF ML, RBF, Kohonen	No	13b	8	Off chip	32 MCPS
			Hybrid Implementations			
AT&T ANNA	FF ML	No	3b×6b	16–256	4096	2.1 GCPS
Mesa Research Neuralclassifier	FF ML	No	6b×5b	6	426	21 GCPS
Ricoh RN-200	FF ML	BP	N/A	16	256	3.0 GCPS

[a] FF ML — Feedforward Multilayer networks; BP — Backpropagation; ROI — Region of Influence
[b] The average speed performance of the retrieve (or feed-forward) process of a neural network model is measured in *million connections per second (MCPS)*
[c] The average speed performance of the learning process of a neural network model is measured in *million connection updates per second (MCUPS)*

Blayo and Hurat [20] presented a Wafer Scale Integration (WSI) neural network dedicated to pattern recognition on associative memory. The presented device consists of implementing the N-neuron Hopfield Network as a systolic square array made up of N^2 cells.

Graf and Henderson [86] designed an analog CMOS neural net with a programmable architecture containing 32k connections with analog signals inside the network and digital signals for all others. The deployed network consists of building blocks that can be joined to form various network architectures and thus can be programmed to implement single-layer networks or multi-layer networks. The chip was fabricated in a $0.9\mu m$ CMOS technology and executed the feed-forward computations within 100ns.

Satyanarayana *et al.* [249] presented the design and implementation of a neural network with programmable topology and programmable weights, built using analog CMOS VLSI technology. They proposed a new "distributed neuron-synapse" circuit block and a array of switches in the interconnections between synapses and neurons to change the network topology. Thus the proposed hardware neural network was able to alter the topology while solving a program, switch off unused synapses, increase the resolution by providing some redundant synapses, and correct offsets commonly observed in analog circuits.

Cox and Blanz [50, 51] firstly moved the neural network implementations to Field-Programmable Gate Array hardware domain. They presented the implementation of GANGLION, a fully interconnected, digital, feed forward connectionist classifier with one hidden layer capable of 4.48 billion interconnections per second. The entire architecture was built using Xilinx XC3090 and XC3042 Logic Cell Arrays (LCAs), which contain 320 CLBs and 144 CLBs respectively and together compute the scaled weighted sum of their fourteen inputs passed through the activation function in the PROM.

Due to the limited computational capability of FPGA devices at that time, Botros and Abdul-Aziz [22, 23] then expanded the hardware implementation of a fully digital MLP using FPGAs to a even bigger scale, where each node (neuron) in the network was implemented with two Xilinx XC3042 FPGAs and a 1K×8b EPROM. This three-layer network (5–4–2) was trained off-line on a PC and the final values of weights were obtained at the end of training session. All internal multiplications and sigmoid activation functions were realized in a look-up table fashion by programming the CLBs and the EPROM.

So far, we have already known a lot of pioneers in the hardware implementations of neural network. However, sometimes the specific hardware has been heavily influenced by the need to address a diverse range of applications and varying demands require a variety of solutions. In order to give a comprehensive evaluation on aforementioned popular implementations from a comparison perspective, Morgan *et al.* [191] analyzed various implementation technologies (FGPAs, VLSI and WSI) in terms of HyperNet system cost, complexity and performance. The HyperNet was a probabilistic hypercube-based artificial neural network proposed by Gurney [91]. Among all three designs using FPGA (Xilinx 4025), VLSI (0.7μm CMOS chip of 180mm^2), and WSI (0.7μm chip of 12.5cm diameter), Morgan *et al.* demonstrated FPGA's promising performance advantage over software implementation at a low cost, as well as the higher performance offered by VLSI and WSI but with high initial development costs.

2.2 IMPLEMENTATION STRATEGIES AND EXAMPLES OF NEURAL NETWORKS ON FPGAS

An artificial neural network (ANN) is essentially a parallel and distributed network of simple nonlinear processing units interconnected in a layered topology [312]. *Parallelism*, *modularity*, and *dynamic adaptation* are three most noticeable and important computational characteristics associated with ANNs. Fortunately, the inherent regularity, homogeneity and reconfigurability of FPGAs makes it a perfect candidate platform to implement ANNs, since it is able to quickly reconfigure itself to adapt any changes in the internal parameters and overall behaviors of an ANN. However, FPGA realization of ANNs with a large number of neurons is still a very challenging task because ANNs are computationally intensive algorithms and it is extremely expensive to exhaustively implement every computational module (adders, multipliers, sum of squares, sum of products, etc.) in each neuron unit. Zhu and Sutton [312] provided a brief survey of existing ANN implementations on FPGAs and re-examined all design issues that are important for such type of implementations. Similar issues were also reported by Hu *et al.* [118] and Muthuramalingam *et al.* [197], including data representation, inner-products computation, implementation of activation functions, storage and update of weights, nature of learning algorithms, serial/parallel design choices and physical design constraints.

Motivated by the increasing demands to provide a complete view on the comparative performance of up-to-date hardware implementations for ANNs, Sun [268] conducted a comparative study on several prevailing ANN solutions for a real biomedical application — cardiovascular disease detection based on electrocardiogram (ECG) analysis. A 51-30-12 three-layer Multilayer Perceptron (MLP) neural network has been developed on a cell phone with Windows Mobile OS, and implemented on an FPGA board and an ASIC chip respectively. The ANN system that runs on the cell phone is a common software program similar as other ANN simulators used for PCs. In comparison, the same ANN system was deployed on two hardware platforms respectively: one is a Xilinx's latest programmable Virtex-5 FPGA board and the other is a fully customized integrated circuit chip using 45nm technology. Given the same training and testing data set, these three implementations showed considerably distinct performance levels in speed and power consumption, as shown in Table 2. Although the performance data is not completely accurate due to experimental tolerance and the relatively small-scale computation, it is shown that the application-specific hardware implementations of ANN systems provide significantly better performance over the conventional software simulations by orders of magnitude. It is desired to explore more effective and efficient hardware solutions to address the future computational challenges of large-scale neural network modeling and simulation.

1. **Feedforward Neural Network**

   Feedforward back-propagation neural network has been widely applied in so many fields such as adaptive control, robotics, and fuzzy computers. Ruan *et al.* [239] described a real-time FPGA-based system used for soft-measuring fields. Using Altera APEX 20k FPGA, the whole system was controlled by a microprocessor chip which configures the FPGA switching between BP training and feed-forward computation by loading certain bitstreams from external memory. Given their previously developed ANN blocks library [281] that can be configured by the designers, Oniga *et al.* presented an implementation of Feed-Forward ANNs with one or two layers in a modular construction fashion, used for smart devices that needs learning capability and adaptive behavior [207]. They also

Table 2: Comparison of Performance and Power Consumption among Software, FPGA, and ASIC Implementations of An ANN System

Properties		**Embedded Software**[a]	**FPGA**[b]	**ASIC**[c]
Performance	One Propagation (us)	13500	60	2.5
	Training Set (hours)	2700	12	0.5
Power	Operation (W)	0.44	1.2	0.6[d]

[a] The program runs on an AMOI A85 with Fone+ Base cellphone with Microsoft Windows Mobile 5 OS.
[b] The ANN system runs at 100MHz on a Xilinx Virtex-5 XC5VLX110T FPGA.
[c] The chip is designed and simulated using NCSU FreePDK for 45nm technology.
[d] The power consumption is assessed without memory.

presented an in-depth study particularly on the error reduction as a function of number of bits used for weight representation, the influence on resource occupation of Xilinx block parameters and the potential maximum working frequency [206]. Other implementation efforts of feed-forward neural networks on FPGAs also include [68, 84].

2. **Radial Basis Function (RBF) Network**

Radial Basis Functions (RBF) networks are powerful tools for interpolation in multi-dimensional space and have been known to learn data by measuring the Euclidean distance with the advantage of not suffering from local minima as MLP. Kim and Jung [144, 145] presented and evaluated the hardware implementation of an RBF network whose internal weights were updated in the real-time fashion by the back-propagation (BP) algorithm. Different from prior designs, a dedicated floating-point processor was designed on an FPGA to execute nonlinear functions required in the parallel calculation of the BP algorithm. Similarly, a RBF-DDA (Dynamic Decay Adjustment) neural network was implemented by Aberbour and Mehrez [1], which was primarily used in the classification of image signatures extracted from gray-level images. Moreover, in the study conducted by Krid *et al.* [151], both a back-propagation feed-forward neural network (BFNN) and an RBF neural network (RBFNN) were implemented on a Xilinx

Virtex-II FPGA. The performance comparison provided a new insight on the quantization effects, which is the RBFNN had a particular sensitivity to quantization errors and thus required more hidden layers to obtain acceptable errors in the network outputs.

3. **Kohonen Self-Organizing Network**

   Self-Organizing Map (SOM) [147, 148] is an unsupervised neural network with competitive learning models that can capture the topology and probability distribution of input data, which has been widely used in pattern recognition for clustering and classification. A binary SOM was designed and implemented on a Xilinx Virtex-4 FPGA by Appiah *et al.* [11], the learning algorithm of which maintained and updated tri-state vector weights to facilitate network training process. A similar binary weighted vector SOM was proposed and simulated in Yamakawa *et al.*'s study [304] to entirely avoid numeric weights in the SOM while maintaining considerable levels of performance and speedup for real-time applications. The Hamming distance was used to calculate the distance between the input and weight vectors, to identify the winning neuron in the network. Two other SOM implementations with simplified computations of the distance, neighborhood and learning rate were presented by Chang *et al.* [32] and Porrmann *et al.* [226].

4. **Recurrent Network**

   Recurrent Neural Networks (RNNs) have interesting properties and can handle dynamic information processing unlike ordinary feedforward neural networks. However, they are generally difficult to use because of no convenient learning scheme available and thus the difficulty in setting up the values of the weights in the network for specific purposes. Maeda *et al.* [179, 180, 181] proposed a recursive learning scheme for recurrent neural networks using the simultaneous perturbation method, which is also applicable to analog learning and the learning of oscillatory solutions of RNNs. As a typical recurrent neural network, Hopfield Neural Network [109] with symmetrical fully connected weights has attracted a great deal of interest and been widely used to store patterns and solve combinational optimization problems. Based on the aforementioned simultaneous perturbation learning rule, Maeda *et al.* [178, 290] implemented the Hopfield neural networks with learning capability on an Altera EP20K FPGA. Saif *et al.* [242] presented an FPGA-based implementation of a Competitive Hopfield Neural Network (CHNN) to accelerate image processing, which are usually believed to be computationally expensive and even more time consuming as the amount of input data increases. Similarly, Abramson *et al.* [2] discussed the implementation of HNNs for solving constraint satisfaction problems using FPGAs to achieve a speedup of up to 3 orders of magnitude.

5. **Probabilistic Neural Network**

   Probabilistic Neural Network (PNN) [264] (also known as "Stochastic Neural Network"), is one of the statistical pattern recognition techniques and built by introducing random variations into the the network. PNNs can be used to solve statistical pattern recognition problems based on the Bayesian discrimination theorem [258]. Minchin and Zuknich [185] proposed to reduce the complexity and memory consumption of PNN and make it possible to implement PNN in standard FPGA logic devices, by developing a low computationally complex hardware design based on fixed-point binary vector components as well as simple spherical basis functions and distance measures. Due to its capability in attaining higher accuracy, PNNs have been widely applied to the pattern discrimination problems for bioelectric signals [264]. Shima and Tsuji [258] proposed a new PNN architecture using delta-sigma modulation (DS modulation) to realize the high performance EMG pattern discrimination. They then implemented the proposed PNN on the FPGA and reported a 2.13% decrease of digital circuit scales and a 12% increase of calculation speed. Mizuno *et al.* [187] reported a reconfigurable architecture for PNN and developed the PNN hardware system using FPGAs, in which the preprocessing circuits can be reconfigured.

6. **Spiking Neural Network**

   Spiking Neural Networks (SNNs) incorporates the concept of time into their operating model, in addition to neuronal and synaptic states. SNNs generate behaviors and reproduce coding schemes closely analogous to biological neural systems [177] and are consequently used extensively to model the operational functionality of the brain, such the Blue Gene supercomputer launched by IBM [129]. Pearson *et al.* [211, 212] presented the implementation of a large scale (over 1000 neurons), leaky-integrate-and-fire neural network processor using the Xilinx Virtex-II FPGA. Harkin *et al.* [99] discussed the challenges of implementing large scale SNNs on reconfigurable FPGAs and presented a novel Field Programmable Neural Network (FPNN) architecture incorporating low power analogue synapse and a network-on-chip architecture for SNN routing and configuration. Shayani *et al.* [256] proposed a digital neuron design with a novel flexible dendrite architecture and the new PLAQIF (Piecewise-Linear Approximation of Quadratic Integrate and Fire) soma model, and implemented a network of 161 neurons and 1610 synapses on a Virtex-5 FPGA. Another preliminary investigation regarding the FPGA-based SNN implementation for a tangible Collaborative Autonomous Agent was conducted by Bellis *et al.* [16]. From a perspective of architectural acceleration, Hellmich and Klar [105] described an FPGA based simulation acceleration platform and demonstrated the acceleration factors of 4 to 8 for computationally intensive numerical integration part.

Perhaps the greatest advantage of ANNs is their ability to be used as an arbitrary function approximation mechanism which 'learns' from observed data. This is particularly useful in applications where the complexity of the data or task makes the design of such a function by hand impractical. As shown above, a large family of most frequently used neural networks have been deployed and implemented on custom reconfigurable hardware — FPGAs, as one of the efforts to achieve the most flexible and efficient neural networks with more considerable performance than the software simulations. Furthermore, a large number of significant contributions on the FPGA-based application specific neural network architectures have been reported by researchers, covering broad categories such as image and video processing [150, 161, 190], audio processing [128], industrial automatic control [137, 146, 152, 205, 274, 310], medical applications [6, 66].

2.3 FAULT-TOLERANT ARTIFICIAL NEURAL NETWORKS

Artificial neural networks are inspired by natural neural networks in the human brain and consist of distributed processing elements with each node contributing to the final output response. The human brain exhibits a remarkable degree of fault tolerance since it continues to function in spite of losing as many as 10^4 neurons per day [57]. Fault tolerance is therefore a desirable property and is believed to be an intrinsic property of ANNs. The main reason advanced for this belief is the fact that the storage mechanism is *connectionist* and cutting off a few neurons and their associated interconnections presumably should not affect the performance of the network drastically.

There are generally several distinct strategies to improve the fault tolerance of a neural network. In a straightforward manner, the first strategy is to explore the inherent fault tolerant characteristics of neural networks. Chun [45] firstly provided a thorough analysis on the inherent fault tolerance characteristics of neural networks. He and McNamee [46] then proposed a method that models the effects of fault in an ANN as deviation in weight values after the neural network has been trained. A fault in circuit where all weight information are stored may cause a stored value destroyed or retrieved incorrectly. Based on this model, they proposed to use fault injection to improve fault tolerance and reduce the sensitivity of a neural network's output to changes in weight values.

Séquin and Clay used a stuck-at fault model to describe the effects of faults that can occur in weights and units of neural networks [254]. That is, neural network output (or a weight) is stuck at the maximum/minimum value or a value between them. Séquin and Clay have primarily focused on the methods of intendedly injecting emulated faults into neural networks to improve their fault tolerance. To achieve true fault tolerance, in their scheme, hidden units are randomly 'disabled' for some pattern presentations during a standard backpropagation training phase [253]. They claimed that such prolonged training can achieve fault tolerance even with respect to fault patterns for which the network was not trained specifically. They then extended their injected faults and randomly introduced the types of failures that one might expect to occur during operations to develop a more robust neural network [48].

Different from previously introduced schemes devised to tolerate a particular type of fault at a time (e.g., stuck-at-(-1) or stuck-at-1), Arad and El-Amawy [12] described a robust fault tolerant training algorithm that took into account the effect of all possible faulty neurons during each weight updating process and demonstrated that the proposed method can tolerate any single faulty hidden neuron stuck at any value between -1 and +1.

The second strategy is to enhance the fault tolerance of neural networks by investigating appropriate learning algorithms. Horita *et al.* proposed a "deep learning method" for making multilayer neural networks fault-tolerant to multiple weight-and-neuron faults [111] and then implemented such method using VHDL with quantized weights and step activation function [110]. Yamamori *et al.* [305] proposed an efficient built-in fault-tolerant mechanism for ANNs implemented on a digital VLSI chip, named "Partial Retraining", which is applied to only a single neuron affected by the hardware fault instead of the entire multilayer network. Similarly, Kim *et al.* [143] also presented a *Partial Re-learning* scheme to achieve fault-tolerance and accelerate the execution speed, which is applied to only a single neuron level, not entire networks. For the performance criterion used in the training process, Hsieh and Sher [115] defined a term called *constraint energy* and incorporated it with normal energy to control the fault tolerant property of neural network and guarantee some degree of fault tolerance when any one of hidden node failure.

The third strategy of augmenting fault tolerance of neural networks is to simplify the network structure and properly manipulate the synaptic weights within ANNs. It has been well studied and agreed that strong connections make neural networks more sensitive to faults. Thus, a variety of optimization approaches have been proposed to mainly minimize the weights and the number of neuron nodes. Some of representative methods include: partially weight minimization [101], weight minimization [270, 271], dynamic constructive algorithm with minimal number of hidden units [96, 100], replacing summation with median neuron input function [240], and gradient manipulations of activation function [273] for removing nodes that do not significantly affect the network and add new nodes that share the load of the more critical nodes in the network [42].

A network trained by the backpropagation algorithm may not distribute the solution across all the weights [21], which means, some of the weights in the network are indeed critical and the loss of these can cause the network to fail. Deodhare *et al.* [57] claimed that an multi-layer perceptron neural network can exhibit fault tolerance if the information content of the network that captured in the connection weights is uniformly distributed. Thus Neti *et al.* [199] and Deodhare *et al.* [57] formulated the fault tolerance exploration as a *constrained minimax optimization* problem, and attempted to minimize the maximum deviation from the desired output for each input in the presence of signal unit failures. Kamiura *et al.* [138] presented approaches for Hopfield neural networks to tolerate weight faults, including weight restriction and fault injection. The weight restriction determined a range to which values of weights should belong during the training; while a status of a fault occurring is then evoked by the fault injection under which weights were calculated. Concerned about the fault tolerance capability against the weight perturbation, Elsimary *et al.* [71] described a measure criterion that is the deviation of the network's output after training, when each interconnection weight is perturbed, from that output without perturbation, and then tried to maintain that deviation as minimum as possible.

The last strategy, also the most intuitive way, is to use well established traditional fault-tolerance techniques, include spatial redundancy, temporal redundancy, and coding. From coding theory perspective, Petsche and Dickinson [61, 214] explored relationships between neural networks and convolutional or trellis codes that can lead to fault tolerant behavior. The authors claimed that convolutional and trellis codes are of interest to neural network researchers because they can lead to some types of coarse coded or receptive field representations, as well as these trellis-structured networks can detect and correct errors in the inputs in a well-behaved way. Ito and Yagi [127] proposed a new fault tolerant multilayer neural network which can correct an error caused by a fault in the output layer. The underlying idea is to use an error correcting code for NN's output space and let NN learn this code in the training phase.

Phatak *et al.* related fault tolerance to the amount redundancy required to achieve it and demonstrated that less than TMR (Triple Modular Redundancy) is not sufficient to achieve complete fault tolerance for the standard ANN architectures [216, 218, 219]. He then proposed an simple alternative method of replicating a seed network, to enhance partial fault tolerance (PFT) of ANNs [216, 218]. Unfortunately, it is shown that this method still requires a large amount of redundancy even if the size of the seed network (which get replicated) is kept minimal [216, 217] and a brute force method of replications seems to achieve a higher PFT for the same level of redundancy as compared with the gradient descent training [218, 219]. Therefore, Phatak and Tchernev [220] presented and investigated an improved approach to obtain the optimal size of the seed network that achieves the highest PFT for a fixed final size (i.e., the total number of units and connections). Concerned about the area and time overhead caused by conventional TMR method, Ahmadi *et al.* [4] demonstrated a fault-tolerant implementation of neural networks, which only contains one extra neuron in the hidden layer and can correct any single fault with less than 40% area overhead.

2.4 RECONFIGURABLE ARTIFICIAL NEURAL NETWORKS

Traditional fault-tolerance strategies include spatial redundancy, temporal redundancy, and coding [227]. A typical technique of spatial redundancy is the triple modular redundancy (TMR). Chu and Wah [44] proposed and analyzed a robust fault-tolerant neural network by inserting hybrid redundancy (i.e., a combination of spatial redundancy, temporal redundancy, and coding) on the output layer neurons. In their approach, every result if computed m times each by different neurons for voting, which leads to a tremendous temporal overhead. On the other hand, there are m copies of storage banks in each output neuron N_i, where each bank stores the weights associated to the connections incident on neuron N_i and its neighboring neurons of the same layer. Thus this approach may also induce large amount of area overhead, since it tends to have large number of input neurons in a neural network. Based on Chu's work, Chen *et al.* [38] presented a unified reconfigurable fault-tolerant multi-layer feed-forward neural network to address both fault detection and reconfiguration of a neural network. Basically, in Chen's approach, a concurrent error detection scheme is integrated into the storage bank of each neuron to detect any fault in the data array of the storage bank leveraging extra parity bits, as well as a reconfiguration scheme with many identical spare neurons is introduced to the output layer of the neural network, where the outputs from either regular neurons or spare neurons are selected by a dedicated receiver. Chen's work is probably the first research effort to address the reliability of neural networks from a reconfigurable hardware perspective.

Motivated by the reliable operation of large scale neural networks implemented into a large chip or silicon wafer, Sugawara *et al.* [266] proposed a fault tolerant multi-layer neural networks employing both hardware redundancy and weight retraining in order to realize self-recovering neural network. Involving a few spare neurons in each layer, a selector, placed between each, chose one output of a selected neuron in the previous layer and inputted it into all neurons in the next layer simultaneously. In this way, the swapping between a faulty neuron and a spare neuron can be easily achieved by the associated selector. Unfortunately, this system exploited the genetic algorithm (GA) to train the neural network and generate corresponding synaptic weights with dedicated external processor and memory. Unlike the prevailing back-propagation training algorithm, such GA-based neural network only needs to maintain a weight lookup table and no computation has been involved in each neuron node at all.

Analog circuit techniques provide area-efficient implementations of the functions required in a neural network, such as multiplication, summation and the sigmoid transfer characteristics [87, 247]. However, they are prone to problems like offsets and gain errors due to mismatches in identically designed devices and inaccuracies in device models. To provide a programmable topology as well as the programmability of weights, Satyanarayana *et al.* [246, 248, 249] presented a reconfigurable neural network on a VLSI chip featuring 1024 “distributed neuron-synapse”. Using switches in the interconnections between synapses and neurons permits one to change the network topology. Unfortunately, such reconfigurable neural network design is based on a relatively straightforward gradient-based learning algorithm, rather than the well established back-propagation algorithm. As the authors mentioned, this algorithm does not require an analytical expression for the sigmoid and its derivative, and thus it can be easily adapted for training networks with different topologies. The topological reconfigurability was established on the basis of the identity and simplicity and neuron units. Similar reconfigurable neural network designs based on programmable inter-connectionism or topology scalability have been presented by many research efforts. Unfortunately, all of these designs were still established based on a relatively simple neural network infrastructure, that is, either an external training processor is needed [308] or no learning capability has been reported at all [68, 90, 225, 280, 311].

FPGA-based designs in all markets are beginning to incorporate more and more features, missions, and waveforms. Among all prior efforts in reconfigurable neural network area, Eldredge and Hutchings [69, 70] firstly described the Run-Time Reconfiguration Artificial Neural Network (RRANN), that uses run-time reconfiguration to increase the number of hardware neurons implemented on a single Xilinx XC3090 FPGA. The key idea of RRANN is to divide the backpropagation algorithm into the sequential execution of three stages known as feed-forward, back-propagation, and update, and further configure the FPGAs to execute only one stage at a time. When one stage completes, the FPGAs are reconfigured with the next stage. This process of "configure and execute" is repeated until the algorithm has completed its task. The RRANN was built on a FPGA board connected to the host PC, which stores all configuration information for the FPGAs, monitors the progress of each stage of execution, and supplies the appropriate configuration data to the FPGA board. However, not surprisingly, such reconfiguration built in a off-line fashion caused significant reconfiguration time cost at approximately 30ms, about 5 times of the minimum configuration time for a XC3090 of approximately 7ms. Beuchat and Haenni *et al.* [18, 92] designed a network computer — RENCO (REconfigurable Network COmputer), that contains a reconfigurable part composed of four Altera Flex10K FPGAs. Based on Eldredge's approach, the described system was also divided into several sequentially executed stages, each of which was associated with a peculiar FPGA configuration; and thus RENCO is reconfigured during the network training. As the first effort to design a size-adjustable neuron network, Pérez-Uribe and Sanchez [213] proposed the FAST (Flexible Adaptable-Size Topology) architecture, a neural network that dynamically adapts its size, and described its implementation on a FPGA. Although the proposed FAST architecture is capable of adjusting the network's size by adding a new neuron in the output layer when a sufficiently distinct input vector is encountered, it is still an algorithmic improvement and incremental modification on the conventional artificial neural network, rather than the real 'physically' dynamic reconfiguration on the whole network structure.

Partial Dynamic Reconfiguration (PDR) [176] is an emerging feature supported by modern FPGAs allowing specific regions of an FPGA to be reconfigured on the fly. Inspired by the evolution, development and learning processes in living beings, Upegui *et al.* have concentrated on the development of *Evolvable Hardware* using state-of-the-art reconfigurable platforms [284] and presented a reconfigurable spiking neural network with Hebbian learning [285, 286]. Essentially, they compiled different partial bitstreams implementing layer topologies available for each one of the modules. Then, from a repository of layers, an evolutionary algorithm will determine the set of layers most adequate for solving the problem.

All aforementioned studies presented attractive research findings on the flexibility augmented and computationally efficient artificial neural network implementations, leveraging various hardware implementation techniques. Unfortunately, none of them has been proposed to effectively and efficiently address the reliability issues of neural networks, especially for the back-propagation learning enabled ANNs, from neuron-level system adaptation perspective. One drawback of previous studies is the redundancy (i.e., space redundancy or time redundancy) introduced to completely guarantee the recovery of neural networks. For instance, Chen's method [38] inevitably involves a large amount of hardware redundancy, caused by the added spare neurons and duplicate weight information stored in spare neurons in case their neighbor nodes are damaged. Another major constraint of prior reconfigurable neural network studies is their limited applicability without the well-established back-propagation learning capability. It is well agreed that the back-propagation learning algorithm involves complicated operations, which make neural network prohibitive to the implementation on digital hardware, not to mention the desired reconfiguration capability. Sugawara's design [266] used a GA-based learning algorithm and thus no computation has been involved on neuron nodes at all. Satyanarayana's neural chip [246, 248, 249] employed the gradient-based learning algorithm, which does not require extra operations for neuron nodes besides the multiplication and addition. The identity and simplicity and neuron units make them applicable for any locations within the network and facilitate the system topological reconfiguration by simply changing the switches in the interconnections between synapses and neurons. Upegui's reconfigurable spiking neural network [285, 286] is fundamentally a module-based topology optimization problem and trained by the Hebbian learning algorithm. It is not quite suitable for the manipulation of individual neurons to react unexpected neuron faults. Lastly, some relevant higher-level reconfigurable neural network platforms were developed to meet certain performance goal, rather than a finer-grained reconfiguration on the basis of neuron units. For example, Eldredge's FPGA-based reconfigurable ANN platform [69, 70] was proposed to meet the implementation issues caused by the limited logic resources available on then FPGAs.

Given all the constraints and limitations of previously demonstrated reconfigurable neural network systems, it is shown that making the high structured neural networks truly reconfigurable at the neuron level is extremely difficult. Particularly, considering the highly involved back-propagation learning algorithm, it is even more challenging to adapt the topological structure of neural networks during the training process. Inspired by the precise, systematic, sophisticated, and essentially autonomous reconfiguration-based automatic recovery mechanisms of the mammalian Central Nervous System (CNS) to react to unexpected injuries or diseases, we would like to explore the possibility of mimicking CNS's faulty reaction strategies to address the reliability issues of artificial neural network systems. The thesis engineer a novel autonomously reconfigurable artificial neural network architectural framework, that is capable of enhancing the adaptivity, flexibility, reconfigurability, efficiency, reliability, and particular autonomous ability of ANN architectures by integrating innovative architectural solutions and sophisticated hardware design methodologies.

3.0 ARTIFICIAL NEURAL NETWORK

We saw in the last chapter that, artificial neural network (ANN) is an established biologically inspired computing paradigm and has proved to be very effective in a variety of real-world problems. Such neural networks usually involve a highly structured network of simple processing elements (neurons), which can exhibit complex global behavior, determined by the synaptic connections between processing elements and specific element parameters. Among all ANN variants, Multilayer Perceptron (MLP) is one of the most effective and efficient solutions capable of modeling highly complex relationships between given sets of input data and a set of appropriate output to explore the underneath correlated patterns in data. Accordingly, some certain learning algorithms are needed to help ANN figure out a set of appropriate internal configuration parameters. However, the computational effort needed for finding the correct combination of weights increases substantially when more parameters and more complicated topologies are considered.

In this chapter, we will briefly introduce the structure characteristics of multilayer perceptron neural networks and then discuss a popular learning method capable of handling such large learning problems — *the back-propagation algorithm.* In order to describe the conceptual process of ANNs from a practical point of view and facilitate the following discussion of the overall behaviors of ANNs, we also present a case study on the effective use of ANN in emerging biomedical applications — an ANN-based model for limb end-point locomotion predictions.

3.1 NEURAL NETWORK THEORY AND MULTILAYER PERCEPTRON

The human brain, which consists of approximately 100 million neurons that are connected by about 100 trillion connections, forms the most complicated object known in the universe [204]. Except for the tremendous data manipulation and processing, the brain undoubtedly outperforms a digit computer in many other intelligent domains, such as the pattern recognition, sensory information cognition and processing, and the most attractive capability of self-adaptive learning. All such computations of the brain are attributed to that highly interconnected massive neural network, which communicate by sending electric pulses through the neural wiring consisting of axons, synapses and dendrites. In 1943, McCulloch and Pitts [183] modeled a neuron as a switch that receives input from other neurons and, depending on the total weighted input, is either activated or remains inactive. The weight, by which an input from another cell is multiplied, corresponds to the strength of a synapse — the neural contacts between nerve cells. It was then shown that networks of such model neurons have properties similar to the brain: they can perform sophisticated pattern recognition, and they can perform even if some of the neurons are destroyed.

Inspired by the early models of sensory processing of the brain, an Artificial Neural Network (ANN) can be created by simulating a network of model neurons in a computer. By applying algorithms that mimic the processes of real neurons, we can make the network "learn" to solve many types of problems [154]. ANNs possess many attractive characteristics that may ultimately surpass some of the limitations in classical computational systems. The processing in the brain is mainly parallel and distributed: the information are stored in connections, mostly in myeline layers of axons of neurons, and hence, distributed over the network and processed in a large number of neurons in parallel. The brain is adaptive from its birth to its complete death and learns from exemplars as they raise in the external world. Neural networks have the ability to learn the rules describing training data and, from previously learned information, respond to novel patterns. ANNs are fault-tolerant, in the sense that the loss of a few neurons or connections does not significantly affect their behaviors, as the information processing involves a large number of neurons and connections. ANNs have been widely and massively used to model complex relationships between inputs and

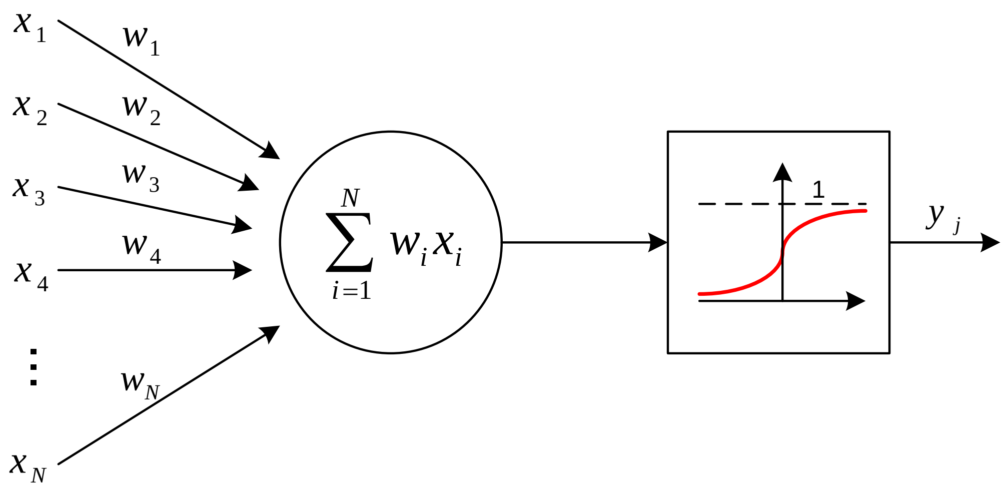


Figure 1: An Artificial Neuron Based on McCulloch-Pitts Model

outputs in many domains, e.g., function approximation, classification (pattern recognition and decision making), robotics, system identification and control, medical diagnosis, data mining, and financial applications. In what follows, we will describe the basic artificial neuron modeling and a particularly important neural network model — Multilayer Perceptron, since it is the most frequently used as well as the most convenient neural network structure.

An artificial neuron forms the basic unit of artificial neural networks. The basic elements of an artificial neuron are (1) a set of input nodes that receive the corresponding input signals or pattern vectors $\mathbf{X} = (x_1, x_2, \ldots, x_I)^T$; (2) a set of synaptic connections whose strengths are represented by a set of weights $\mathbf{W} = (w_1, w_2, \ldots, w_I)^T$; and (3) an activation function ϕ that relates the total synaptic input to the output (activation) of the neuron. The main components of an artificial neuron is illustrated in Figure 1.

The total synaptic input, u, to the neuron is given by the inner product of the input and weight vectors:

$$u = \sum_{i=1}^{I} w_i x_i \tag{3.1}$$

where we assume that the threshold of the activation is incorporated in the weight vector.

The output activation, y, is then given by

$$y = \phi(u) \tag{3.2}$$

where ϕ denotes the activation function of the neuron. The total synaptic input is then transformed to the output via the non-linear activation function. Consequently, the computation of the inner-products is one of the most important arithmetic operations to be carried out for a hardware implementation of a neural network. This means, besides the conventional individual multiplications and additions, a sequence of multiply-add operations is particularly of interest and significance. We shall see the specific efficient and effective design strategies proposed and elaborated in the Chapter 5 in order to augment the performance of hardware implementation of artificial neural networks by sufficiently utilizing the available resources on modern FPGAs.

The Multilayer Perceptron (MLP) is a feed-forward artificial neural network model which maps sets of input data onto a set of appropriate output. It is a variant of the classic linear perceptron with three or more layers of neurons equipped with nonlinear activation functions. Typically, as shown in Figure 2, a standard MLP consists of an *input layer* of nodes, followed by two or more layers of perceptrons, the last of which is the *output layer* and all others are referred to as *hidden layer* [103]. It is well agreed that the MLP is more powerful than the perceptron in that it can distinguish data that is not linearly separable, or separable by a hyperplane [52]. MLPs have been applied successfully to many complex real-world problems consisting of non-linear decision boundaries, especially for any supervised-learning pattern recognition process and the subject of ongoing research in computational neuroscience and parallel distributed processing. Currently, the most frequently used MLP is three-layer MLPs, which has been reported to be sufficient for most of aforementioned applications.

The key property of MLPs is that each neuron uses a nonlinear activation function which was developed to model the frequency of action potentials or firing of biological neurons in the brain. Such activation functions must always be normalizable and differentiable, and two main ones used in current applications are both sigmoids described as follows:

$$\phi(y_i) = tanh(v_i) \quad \text{and} \quad \phi(y_i) = \frac{1}{1+e^{-x}}$$

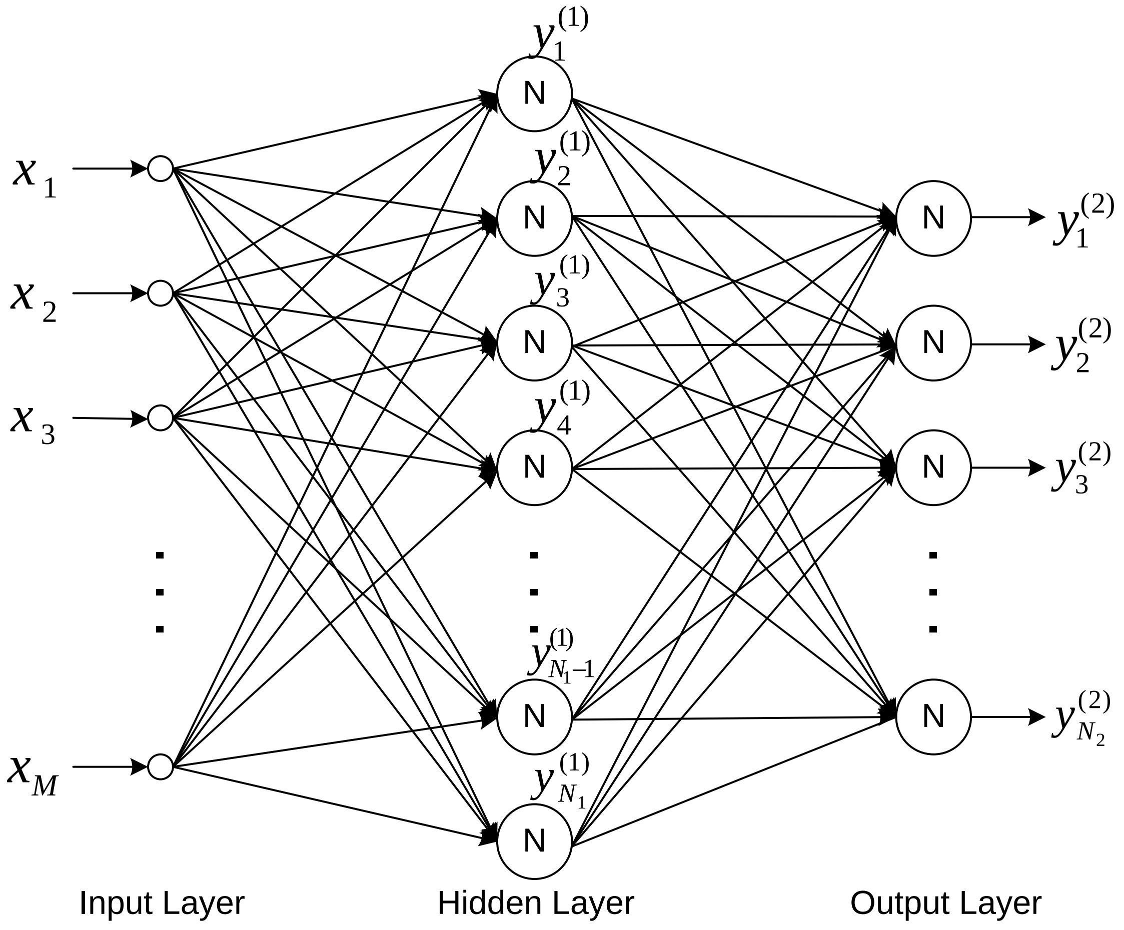


Figure 2: Multilayer Feed-Forward Neural Network

where the former is a *hyperbolic tangent* that ranges from -1 to 1, and the latter is equivalent in shape but ranges from 0 to 1. Here y_i is the output of the ith neuron and v_i is the weighted sum of input synapses.

3.2 BACK-PROPAGATION TRAINING ALGORITHM

Given the feed-forward calculations of desired outputs for any given input in MLP neural network, an appropriate supervised learning method is needed to direct the neural network to achieve a desired training accuracy level by adapting and adjusting its synaptic parameters. The back-propagation algorithm looks for the minimum of the error function in weight space using the method of gradient descent. The combination of weights which minimizes the error function is considered to be a solution of the learning problem. Since this method requires computation of the gradient of the error function at each iteration step, we must guarantee the continuity and differentiability of the error function, which makes this BP process in favor of a kind of continuous activation function rather than the step function [237].

An MLP using the back-propagation algorithm has five basic steps of execution [193]:

1. **Initialization**

   The following parameters must be initialized before training starts: (i) $w_{jk}^{(s)}(n)$ is defined as the *synaptic weight* that corresponds to the connection from the jth neuron in the Layer $(s-1)$ to the kth neuron in the Layer s. This weight is updated during the nth iteration. For the initialization process, the n should be zero. (ii) η is defined as the *learning rate* and is a constant scaling factor used to control the step size in error correction during each iteration of the back-propagation algorithm. (iii) Sometimes, a *momentum factor*, β, need to be introduced, which essentially allows a change to the weights to persist for a number of adjustment cycles (called "*momentum*") in weight adjustment. (iv) $\theta_k^{(s)}$ is defined as the *bias* of a neuron, which is similar to synaptic weight in that it corresponds to a connection to the kth neuron in the Layer s. Statistically, bias can be thought of as noise, which better randomizes initial conditions, and increases the chances of convergence [193].

2. **Presentation of Training Cases**

   Available training data are presented to the network either individually or as a group (called "*epoch*").

3. **Feed-Forward Computation**

   During the feed-forward calculation, data is propagated from neurons in a former layer (e.g., Layer $(s-1)$) forward to neurons in the latter layer (e.g., Layer s), via a feed-forward connection network. The computation performed by each neuron (in the hidden layers) is as follows:

$$H_k^{(s)} = \sum_{j=1}^{N_s-1} w_{jk}^{(s)} o_j^{(s-1)} + \theta_k^{(s)} \tag{3.3}$$

   where $j < k$ and $s = 1, \ldots, M$
   $H_k^{(s)}$ — *weighted sum* of the kth neuron in the Layer s
   $w_{jk}^{(s)}$ — *synaptic weight* as defined above
   $o_j^{((s-1))}$ — *neuron output* of the jth neuron in the Layer $(s-1)$
   $\theta_k^{(s)}$ — *bias* of the kth neuron in the Layer s.

   On the other hand for neurons in the output layer, the computation is as follows:

$$o_k^{(s)} = f(H_k^{(s)}) \tag{3.4}$$

   where $k = 1, \ldots, N$ and $s = 1, \ldots, M$
   $o_k^{(s)}$ — *neuron output* of the kth neuron in the Layer s
   $f(H_k^{(s)})$ — *activation function* applied to the weighted sum $H_k^{(s)}$.

   Note that a unipolar sigmoid function is often used as the nonlinear activation function, such as the following *logsig* function:

$$f(x)_{logsig} = \frac{1}{1+e^{-x}} \tag{3.5}$$

4. **Back-Propagation Training**

In this step, the weights and biases are updated. The goal of learning algorithms is to minimize the error between the expected ("target") value and the actual output value obtained from the Feed-Forward calculation. The following steps are performed:

a. Starting with the output layer, and moving back towards the input layer, calculate the local gradients according to the following equation:

$$\epsilon_k^{(s)} = \begin{cases} t_k - o_k^{(s)} & s = M \\ \sum_{j=1}^{N_s+1} w_{jk}^{(s+1)} \delta_j^{(s+1)} & s = 1, \ldots, M-1 \end{cases} \tag{3.6}$$

where

$\epsilon_k^{(s)}$ — *error term* for the kth neuron in the Layer s; the difference between the target value t_k and the neuron output $o_k^{(s)}$

$\delta_j^{(s+1)}$ — *local gradient* for the jth neuron in the Layer $(s+1)$.

$$\delta_k^{(s)} = \epsilon_k^{(s)} f'(H_k^{(s)}) \qquad s = 1, \ldots, M \tag{3.7}$$

where $f'(H_k^{(s)})$ is the derivative of the activation function.

b. Calculate all weight and bias changes in the following way:

$$\Delta w_{jk}^{(s)} = \eta \delta_k^{(s)} o_j^{(s-1)} \qquad k = 1, \ldots, N_s; j = 1, \ldots, N_{s-1} \tag{3.8}$$

where $\Delta w_{jk}^{(s)}$ is the change in synaptic weight (or bias) corresponding to the gradient of error for the connection from the jth neuron in the Layer $(s-1)$ to the kth neuron in the Layer s.

c. Update all the weights and biases as follows:

$$w_{jk}^{(s)}(n+1) = \beta w_{jk}^{(s)}(n) + \Delta w_{jk}^{(s)}(n) \tag{3.9}$$

where $k = 1, \ldots, N_s$ and $j = 1, \ldots, N_{s-1}$

$w_{jk}^{(s)}(n+1)$ — *updated synaptic weight/bias* to be used in the $(n+1)$th iteration of the Feed-Forward Computation

$w_{jk}^{(s)}(n)$ — *synaptic weight/bias* to be used in the (n)th iteration of the Feed-Forward and Back-Propagation Computations, where $n =$ the current iteration

β — *momentum factor* that allows a change to the weights to persist for a number of adjustment cycles (called "*momentum*") in weight adjustment

$\Delta w_{jk}^{(s)}(n)$ — *changes of synaptic weights/biases* calculated in the (n)th iteration of the Back-Propagation process, where $n =$ the current iteration.

5. **Iteration**

   Reiterate the Feed-Forward Computation and Back-Propagation Training for each training case in the epoch. The MLP will be trained continuously using one or more epochs, until some stopping criteria is met. Once training is complete, the MLP only needs to carry out the Feed-Forward Computation when used in the applications.

3.3 LIMB ENDPOINT LOCOMOTION PREDICTION — AN ANN BIOMEDICAL CASE STUDY

3.3.1 Motivation and Background

Spina bifida (SB) is a developmental birth defect involving the neural tube, usually occurring at the lumbar or sacral levels of the spine. The incidence of SB in the United States is reported between 4 to 10 cases for every 10,000 live births (approximately 70,000 individuals in total), with estimated medical costs of over $200 million [158]. Most of these health care costs are used for physical surgeries, assistive technologies, and rehabilitation to improve patients' walking ability. Due to the notable incidence and costs associated with SB, the development of enhanced rehabilitation and intervention treatments to address the locomotion deficits associated with SB is an important clinical goal.

Non-invasive neuroprostheses, such as Functional Electrical Stimulation (FES), neural implants, and robotic limbs, have been widely investigated as means of correcting the locomotion deficits associated with paralyzed individuals who have suffered from spinal cord injury, spina bifida, or other disabilities, and improving their functional movements. In the studies for primates, direct cortical control of invasive neuroprosthetic devices and robotic arms have been proposed and delivered recently [276, 295]. However, such invasive techniques might damage the central nervous system due to adverse affects caused by physical surgeries. Also, some commercially available solutions, like NESS FES system [262], are based on open-loop control which lacks necessary adaptability and accuracy. Thus, in this study, we propose to investigate the possibility of using measured neuromuscular activities from paralyzed individuals to evaluate and predict their gait behaviors. Differing from other studies whose emphasis is the continuous leg, joint or foot trajectories, we are mostly concentrating on the discrete statistical end-point gait information, e.g., double support time ratio, step length, etc. Compared to traditional continuous trajectory-based control, end-point control is able to provide more walking gait characteristics and accurately describe limb movement behaviors.

As discussed above, the artificial neural network (ANN) is one of the most established machine-learning techniques to synthesize a self-adaptable system. Many previous studies have explored the capability of the ANN in synthesizing a self-adaptive system [95, 154, 166] to recognize complex, unforeseen patterns and has been proven to be effective in the domain of cortical-related hand trajectory prediction [295] and autonomous robotic motion planning [306]. Given ANN's prior successes to allow a machine to adaptively learn to recognize complex, unforeseen patterns, the goal of this study is to investigate the feasibility and practical implementation issues of applying ANN theories to develop an artificial neural network-based (ANN-based) technique for neuroprostheses. Specifically, An ANN-based system model was proposed and implemented in our study [34, 33], where EMG signals collected from six muscles and their co-activation behaviors were used to predicting the end-point locomotion parameters.

The study was conducted based on the laboratory data from twelve individuals with lumbar or sacral level SB (5 females and 7 males; age=14.17 $\pm$ 6.07 years; height=1.46 $\pm$ 0.21 meters; weight=56.31 $\pm$ 28.85 kilograms; body mass index=24.64 $\pm$ 6.47 kg/m^2) [33]. Electromyography (EMG) data collected from the subjects using surface electrodes includes *tibialis anterior* (T), *gastrocnemius* (medial head, G), *soleus* (S), *quadriceps* (rectus femoris, QR; vastus lateralis, QV), and *hamstrings* (biceps femoris, H). These preamplified bipolar EMG electrodes were placed on both legs over aforementioned muscle bellies [35]. The gait events, touchdown and toe-off, were determined via behavior coding. The time of touchdown was at the frame in which any part of the foot contacted the ground at the beginning of the stance phase. The time of toe-off was identified when the foot was off the ground at the beginning of the swing phase. We used touchdown to identify onset of each stride cycle. We collected a total of 144 trials (12 trails for each of 12 participants). Trials with missing markers or without at least two complete steps were excluded from the data analysis; thus, we analyzed a total of 127 trials (5 to 12 trials per participant) of leg neuromuscular activity and end-point parameters. Due to the differences in stride cycle duration among individuals, we normalized the burst duration by the stride cycle duration. We calculated the co-activation indexes [74] for each muscle pair, T and G, T and S, QR and H, QV and H, G and QR, G and QV, S and QR, as well as S and QV. Finally, the input variables included each normalized muscle burst duration and muscle co-activation ratio. The output variables were normalized end-point locomotion parameters (stride length, step width, stance phase ratio, double support phase ratio, step cadence (steps per minute), and stride velocity). Due to the differences in leg length among individuals, the gait parameters related to this factor needed to be normalized by leg length [35, 36].

3.3.2 Proposed ANN-based Model

We propose to implement a multilayer ANN-based model to explore the inherent correlation between the intrinsic impaired neuromuscular activities of people with SB and their extrinsic gait behaviors. Specifically, we adopt a three-layer (input, hidden, and output layers) feed-forward network topology as it is one of the most popular schemes that have been shown to offer a balanced trade-off between prediction accuracy and network complexity. We employed *Levenberg-Marquardt* algorithm [136] as the learning algorithm for our ANN-based model, which is a backpropagation-based algorithm [93, 103] that has been shown to be very effective due to its better time efficiency and higher prediction accuracy [202]. We analyzed data using MATLAB version R2007b and ran statistical regression models for comparison using the MATLAB's Statistics Toolbox 6.2.

Figure 3 illustrates the general workflow of using ANN-based model. It includes three layers: input (neuromuscular activity), hidden, and output (end-point gait parameters) lay-

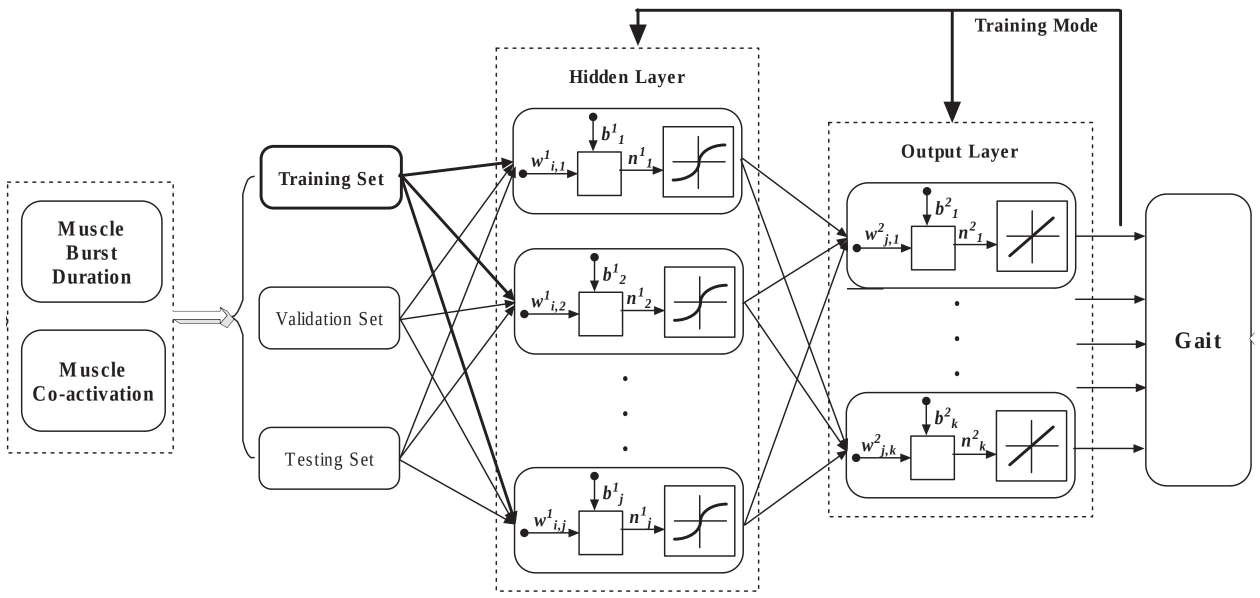


Figure 3: Workflow of The Proposed ANN-Based Technique

ers. The inputs to the ANN are the intrinsic neuromuscular activity. Following the inputs are the hidden and the output layers, where the hyperbolical tangent sigmoid activation function ("*tansig*") and the simple linear activation function ("*purelin*") are used, respectively. Before the ANN-based model can be deployed, it must be trained so that it can learn to recognize the inherent characteristics and complex correlation from the input neuromuscular activity of the target individuals. To start training process, the network weights and biases are initialized randomly. Training the network to produce a desired output vector when presented with an input vector typically involves systematically changing the weights and biases of all neurons until the network produces the desired output within a given error threshold. This tuning process is repeated over the entire training set. Thus, training of an ANN can be simplified to a minimization process of the error measure over the entire training set during a finite number of training cycles.

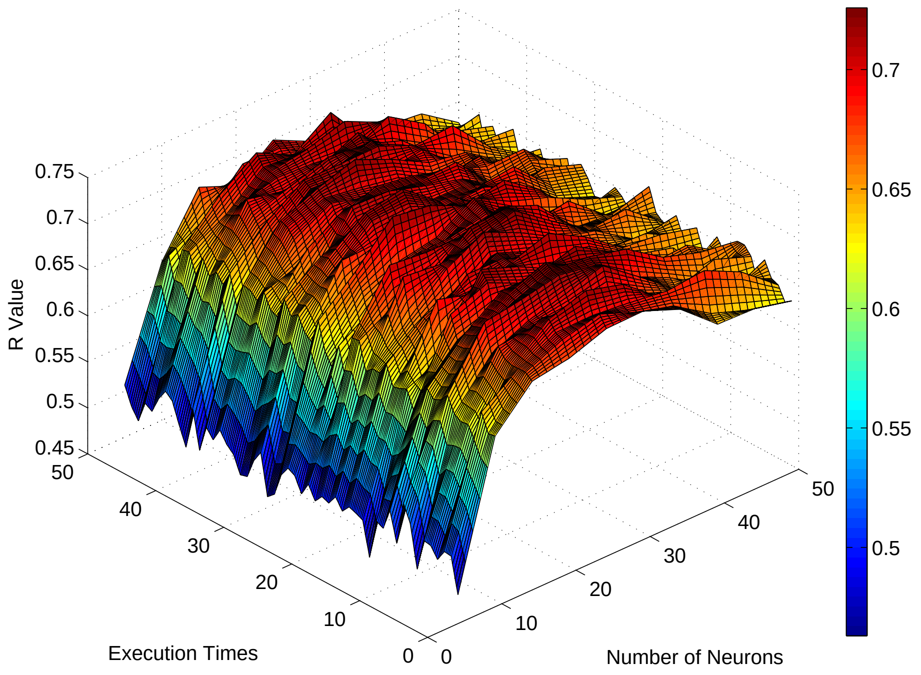
3.3.3 ANN Structure Exploration

In order to identify the most appropriate neural network structure for optimizing its prediction performance, we investigated the fitting accuracy of 10 neural networks, which differed only in the number of hidden neurons (5, 10, 15, 20, 25, 30, 35, 40, 45, 50). Since the hidden layer was where significant portion of ANN learning and the solution processing took place, it was one of the most important parameters that directed the process of the network training and impacted the final fitting accuracy. Thus, we investigated the effect of the number of hidden neurons on prediction performance. To increase the generalizability of this investigation and to avoid the pitfall of drawing conclusions based only on one particular training/validation/testing data set assignment, we independently constructed 50 randomly composed data sets from the sample pool of 127 trails by randomly assigning 2/3 of sample pool to the training set, 1/6 to the validation set, and 1/6 to the testing set. This training/validation/testing set assignment is exclusive: the same trail cannot belong to more than one group. We ran each of the 50 composed data sets on all 10 neural networks and compared their respective fitting results.

We present the fitting performance of all 500 data points (50 compositions of data and 10 neural networks) in terms of the R-values in Figure 4. Our results show that as the number of hidden neurons increased from 5 to 20, the R-values increased substantially; however, as the number of hidden neurons exceeds 30, the R-values gradually decreased. The best fitting results were obtained when the number of hidden neurons was in the range of 20 to 30. Thus, we adopted 25 hidden neurons for our ANN-based model.

3.3.4 End-Point Locomotion Prediction Performance of ANN-Based Model

To demonstrate the efficacy of the proposed ANN-based approach, we compared our ANN-based model with two statistical regression techniques: 1) *Multiple Linear Regression* and 2) *Robust Regression*. Multiple Linear Regression is widely used in statistical analysis, in which the trend exhibited by the observational data is modeled by a linear function that can obtain the best data fitting result. Robust Regression is another linear-like regression technique, which considers the weights of data points and is less sensitive to large changes in



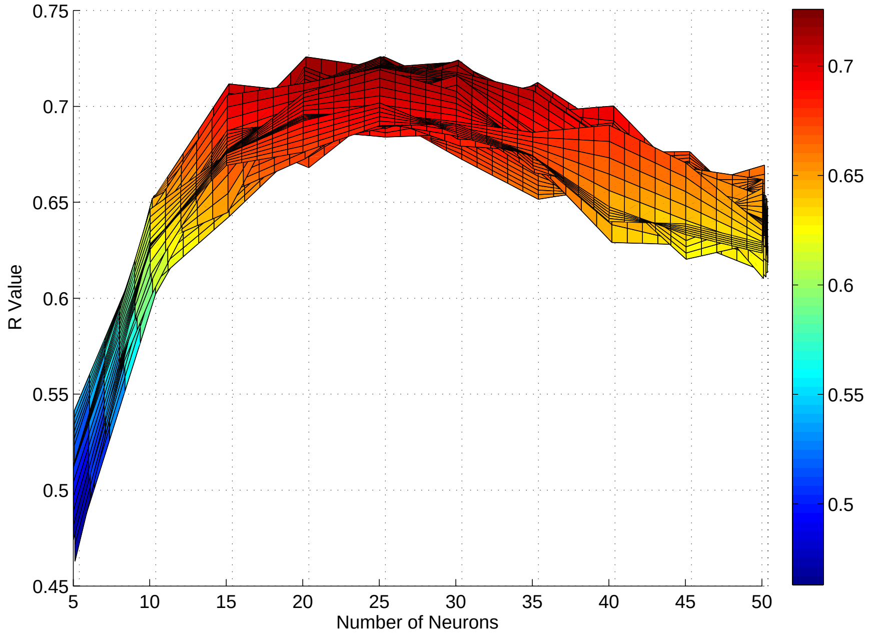


Figure 4: The Accuracy of ANN as A Function of The Number of Hidden Neurons

small parts of the data. In this comparison, we also evaluated another very important ANN parameter: activation function. Hence, there were two different ANN schemes: 1) ANN (tansig+purelin) that used "*tansig*" in the hidden layer and "*purelin*" in the output layer, and 2) ANN (tansig+tansig) that used "*tansig*" activation function in both the hidden layer and the output layer.

To evaluate the fitting performance of all 4 schemes above, we randomly constructed one composition of data set from the sample pool following the procedure described in Section 3.3.3 and designated it as the target composition. Based on the characteristics of the target composition, we trained, optimized, and tested all 4 schemes and obtain their R-values. To evaluate the generalizability of all 4 schemes, we ran them with additional 500 different compositions of data sets randomly generated from the sample pool and obtained the average R-value for each scheme.

We show the fitting performance of the ANN-based and regression-based techniques in Table 3. For single composition of data set, both ANN-based approaches significantly outperformed regression-based approaches in both training and testing phases. For instance, the R-values of ANN (tansig+tansig) were 0.9721 in training and 0.9178 in testing, whereas the highest R-value from both regression-based techniques is no more than 0.6516. Similar conclusions can also apply when the number of data sets was increased to 500 compositions. For instance, the R-values of ANN (tansig+purelin) were 0.9047 in training and 0.7090 in testing, whereas the highest R-value from both regression-based techniques were 0.6601 in training and 0.5385 in testing. We also observed that ANN (tansig+tansig) outperformed ANN (tansig+purelin) for single composition of data sets; and ANN (tansig+purelin) outperformed ANN (tansig+tansig) for 500 compositions of data sets.

To evaluate ANN's prediction accuracy, we compared the predicted values with the actual values for all 6 end-point locomotion parameters, observed from all 12 individuals with SB. We made the comparisons both individually as well as for the group. For group comparison, we trained, validated, and tested ANN-based model using all 127 trails collected from all 12 subjects. For individual comparison, we trained, validated, and tested ANN-based model using the established *Leave-One-Out Cross-Validation* (LOOCV) method to address the issue of smaller data sets [200].

Table 3: Fitting Performance among 2 ANN-based and 2 Statistical Regression-based Prediction Schemes

	Training Set		Testing Set	
	R-value (500 comp.)	R-value (1 comp.)	R-value (500 comp.)	R-value (1 comp.)
ANN (tansig+purelin)	0.9047	0.9595	0.7090	0.8501
ANN (tansig+tansig)	0.8712	0.9721	0.6742	0.9178
Linear Regression	0.6601	0.6264	0.5385	0.6516
Robust Regression	0.6407	0.5566	0.5284	0.5116

We evaluated ANN's prediction power on all 6 end-point locomotion parameters and presented the results in Figure 5. We found that the predicted end-point locomotion parameters were closely matched with their actual observed values. The prediction performance was satisfactory across all 12 subjects, despite the fact the ANN-based model was group-trained. For individually trained ANNs (i.e., one ANN for each subject), we found that their prediction performances were at least as high as the group-trained results.

3.3.5 Remaining Questions

This study developed an ANN-based technique and investigated its feasibility to predict end-point limb motions via intrinsic neuromuscular activity feedback from people with interrupted spinal cord. Our experimental results confirmed our hypothesis that the proposed technique can achieve a highly accurate prediction (e.g., R-values of 0.92 - 0.97, ANN(tansig+tansig) for single composition of data sets). This high prediction accuracy may be due to the fact that we are mainly focusing on predicting end-point gait parameters. Indeed, researchers have proposed adopting end-point prediction as a faster and more accurate strategy for brain-computer interfaces [245]. The benefits of higher speed and accuracy are important features for implementing real-time feedback control for neuroprostheses.

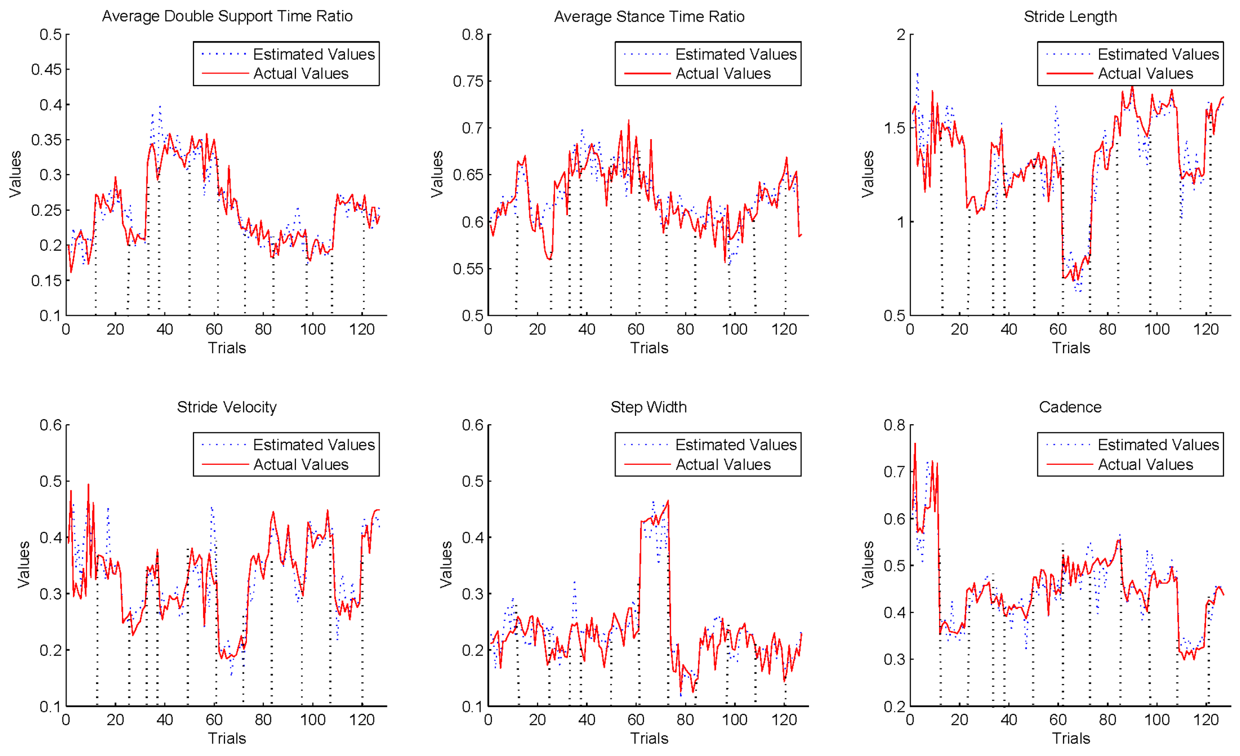


Figure 5: Comparison of The Actual End-Point Locomotion and The Locomotion Predicted by ANN-based Model for All 12 Subjects (Note: The dashed lines delineate the data of one subject from another.)

The results confirmed the hypothesis that ANN-based technique can “learn” to predict end-point motions from neuromuscular activities by recognizing their complex, non-linear relationship. Indeed, we found that ANN-based prediction schemes can consistently outperform regression-based techniques with considerably better (e.g., up to 80% improvement) accuracies. This significantly improved prediction power of ANN-based techniques over the traditional regression-based techniques (as measured in terms of their R-values) can be translated into highly accurate end-point locomotion prediction. Similar successes for ANN-based prediction can also be observed in other problem domains, such as 1) EMG-to-kinematics mapping [39, 40]; 2) cortical responses to auditory spatial perception [303]; 3) cortical neurons in primates-to-hand trajectory mapping [295]; 4) learning behavior prediction [159]; 5) neuromuscular activity generation [229].

Looking ahead, researchers could develop a self-organizing and adaptive controller using low-power, high-performance hardware-software co-design techniques for neuroprostheses to enhance independent movement for people with disability. Given the powerful capability of pattern recognition and system approximation, the implementation of ANNs on state-of-the-art hardware devices, such as ASICs, SoCs, or FPGAs, would significantly advance existing biomedical applications and pioneer new promising techniques/solutions/devices targeting emerging biomedical domains. However, although the hardware implementation of ANNs has shown its considerable advantages and attractive characteristics in processing speed over the traditional software simulation, as well as its promising potential to be developed and deployed for portable biomedical usage, the loss of flexibility would significantly limit their practical applications. Furthermore, the reliability is of great concern to all researchers who advocate ANN specific hardware chips/devices for future medical applications.

Computer systems may fail in any number of ways, such as a fault in the electronic circuit or a bug in the software. To insure that the computer system continues to function in spite of an occasional failure, some certain levels of fault-tolerance capability need to be built into the system or computer. Particularly, for any future ANN-based biomedical portable machines or even implantable devices, the high reliability and some sort of fault-tolerant mechanism are extremely necessary and helpful, because of their relatively difficult system rebuilding or sometimes even unfeasible to recover without a physical invasion.

Engineering fault tolerance into a system generally requires that one replicate a component or process with redundant modules [156]. That is, the designers need to make more than one component available performing the exactly same function. When the fault is detected, the faulty component should be locked and isolated out of the original process and, if necessary, shifting its function to another redundant component. From a higher-level systematic point of view, replicating processors is a straightforward method for contending with a range of system failures. Designers can easily add redundancy and implement fault tolerance with commercial, off-the-shelf processors or devices, without the difficulties of designing application specific redundant components by themselves. Actually, many significant efforts have been made by researchers to build system-level redundancy and fault-tolerant mechanism to prevent any occasional failures or errors. However, there has been little progress made to address the fault-tolerant capability within the system from a micro perspective. Thus, in this study, I would explore the potential reliability requirements on ANN-based hardware platform and work out a more flexible, reliable, fault-tolerant, and self-adaptive ANN platform from a bio-inspired autonomous reconfiguration perspective, leveraging the self-healing and self-optimizing capabilities supported by the proposed architectural innovations.

4.0 AUTONOMOUSLY RECONFIGURABLE ARTIFICIAL NEURAL NETWORK ARCHITECTURE

Chip fabrication facilities have transitioned to 45nm an smaller technologies, resulting in substantial increases in both the number of hard errors, mainly due to variation, material defects, and physical failure during use, as well as the number of soft errors, primarily due to alpha particles from normal radiation decay, from cosmic rays striking the chip, or simply from random noise. Soft errors are not considered to permanently break a circuit; on the other hand a hard error will permanently prevent a circuit from behaving as it was originally designed. It is therefore imperative that chip designers build robust fault-tolerance into computational circuits, and that these designs have the ability to detect and recover the damages causing the system to process improperly or even disabled.

Given the unique computational characteristics discussed in Chapter 3, artificial neural network (ANN) has proved to be effective in a variety of real-world problems [230] and been particularly investigated and advanced for emerging biomedical applications [15, 65, 121, 170]. Many pilot ANN-based medical solutions have been deployed and demonstrated on off-the-shelf hardware platforms [60], including the prevailing programmable logic devices — FPGAs, due to their inherent homogeneity, regularity, and reconfigurability [66, 79, 102]. Unfortunately, the advent of deep sub-micron technology has exacerbated reliability issues on integrated circuits. This spiraling trend highlights the importance of incorporating reliable design methodologies into complex engineered systems, including ANN-based next-generation biomedical systems and implantable devices [58, 59]. Given ANN's promising success in many different areas and increasing demands on ANN-based hardware platforms for emerging biomedical applications, how to implement a reliable ANN-based system and ensure robust operation have been a "hot" research topic of great interest.

Conventional fault-tolerant techniques are available, which can detect, mitigate, and correct many of soft errors and hard faults, such as parity or error correction codes (ECC), dual or triple modular redundancy (TMR), time-redundant computations or checkers [260]. However, these techniques suffer from large overheads that start at 100% and quickly rise if the design must accommodate high availability [184]. More focused fault tolerant techniques have been particularly investigated for ANNs [199, 219]. It was shown that less than TMR is not sufficient to achieve complete fault tolerance of ANNs [277]. Replicating a seed network also requires a large amount of redundancy [219]. Given the fact that all prior error handling techniques applicable to ANN-based systems all need a tremendous amount of redundancy and considerable resource consumption to achieve a high degree of fault tolerance, it is argued that redundancy and complexity alone are not sufficient to guarantee the robust operations of engineered systems. Known for the ability to change a system's structure and operations, Autonomous reconfigurability (AR) has been highlighted (e.g., [26, 198, 209]) as a promising concept for ensuring appropriate operational levels in case of unexpected disturbances.

Inspired by the essentially AR-based self-healing and self-recovery mechanisms of human central nerve systems [275], we try to address the reliability issues of ANN-based systems, particularly focusing on physical damages that influence ANN's overall behaviors and performance, and enhance design space exploration to adaptively find the most effective and efficient solution, leveraging the principles of autonomous reconfigurability. With limited temporal latency and negligible spatial overhead, a reliable and fault-tolerant ANN-based platform is expected to autonomously reconfigure damaged neurons or adapt the structure to tolerate unexpected faults without human interventions. In what follows, we will present the specific architectural innovations of the proposed reliable ANN platform, named "*Autonomously Reconfigurable Artificial Neural Network (ARANN)*".

4.1 SYSTEM OVERVIEW

While there is no precise agreed-upon mathematical definition on the capability of a neural network, it is well agreed that the true power and advantage of neural networks lie in their abilities to represent both linear and non-linear relationships and to learn these relationships directly from the data being modeled. As the most fundamental unit in the ANN structure, each individual processing element — neuron — plays a significant role in the overall system performance. Moreover, the regularity, symmetry, and homogeneity of all interconnected neurons make their synergistic behaviors can be augmented effectively once a design innovation has been applied to each of them. Since the overall behaviors of ANNs are primarily determined by the specific functional characteristics of all homogeneous computational neuron nodes, we will explore the system reliability enhancement solutions from the basic level of the network structure — neuron-level faults, which representing all the damages on the neuron unit causing improper system functionality or even disabling the whole system.

Figure 6 presents the system diagram of the proposed ARANN architecture. ARANN accepts inputs from either existing databases or real-time sensors and then generates corresponding "predicted" results. The *ANN Topological & Algorithmic Controller* is designed to specify the neural network structure and direct the operational workflow of each individual neuron, according to an ANN learning algorithm. Basically, the controller directs the execution of the whole ANN system by updating two variables: the desired number of neurons and the operational instructions to neuron nodes (refer to Section 4.3.1). The *Virtual-to-Physical Neuron Mapping* is a critical component in ARANN, which decouples the "virtual neurons" used in the controller and the "physical neurons" actually deployed on hardware. The mapping connections between virtual and physical neuron ports can be dynamically changed according to controller's demands and physical neurons' availability. For different scales of problems, we propose four possible V2P mapping solutions, which have distinct applicabilities as analyzed in Section 4.5. The error detection module takes charge of monitoring and detecting faults occurring within the neural network (The subject of advanced error detection mechanisms is beyond the scope of this study, please refer to related literature for more details.). Given the specific error scale and location reported by the error detector, the V2P mapping block immediately adapts the mapping connections between virtual neurons and physical neurons, so that the faulty neuron(s) can be disabled and isolated from the main network. Meanwhile, one or more physical neurons will be activated and connected to the network if there are still spare neurons in the neuronal pool.

The neuronal pool contains a number of neuron units physically deployed on the hardware. These neurons are designed and implemented as highly independent, autonomic, and smart computational nodes, which mainly contain a arithmetic core and a private register file storing all relevant information and intermediate results. Given the activation signals, assigned virtual neuron ports, and operational instructions that are originally sent from the ANN controller and then "translated" by the V2P mapping block, all neuron units are able to initiate certain type of operations and return back the acknowledge signal to the ANN controller once they have successfully finished the current operations. It is well known that the standard ANN back-propagation training algorithm is made up of complex training steps and involves extremely data communications among all neuron units through highly structured synaptic connections. Thus, other than the reconfigurable physical infrastructure supporting the ANN structural adaptation on the fly, we proposed a *Dual-Layer Memory Synchronization* mechanism in ARANN architecture, involving a fine-grained synchronization process at each training stage and a coarse-grained synchronization process on the basis of training epoch. Section 4.3.3 presents the proposed dual-layer synchronization mechanism and demonstrates that such mechanism is able to ensure a smooth, accurate, and consistent recovery of neural network systems no matter when a completely unexpected fault is detected. All techniques described above will be elaborated in the following sections.

One more issue worth discussing is the possible error detection (or called "diagnosis") strategies applicable to ANN systems. Fault localization within the neural network is necessary to provide information for hardware reconfiguration in order to achieve system recovery, possibly with reduced computational capabilities (if a considerable amount of logic components have been destroyed). Traditional error detection techniques, which usually involves high overhead in terms of spatial or temporal redundancy, were primarily based on either RE-computing using Duplication With Comparison (REDWC) or the voting results of several replicated components [153]. Hence, an online error detection and retry procedure was considered better to fit resource-/energy-aware portable applications [10].

Concurrent or online error detection (CED) schemes use an output characteristics predictor, which is then compared (using a checker) with actual circuit output to detect an error [228], and thus allow to guarantee continuous checking of results. Concurrent diagnosis is

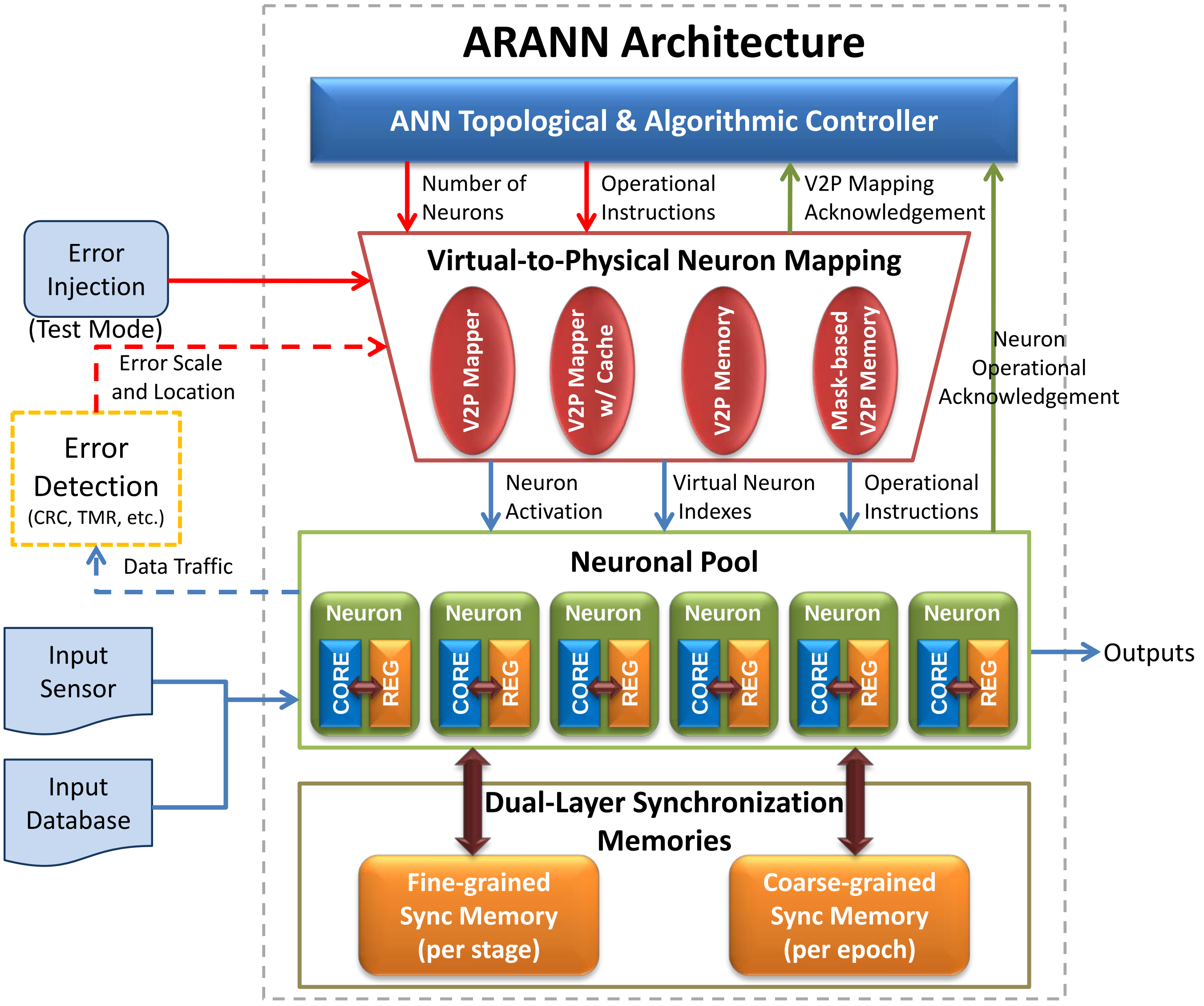


Figure 6: System Diagram of the ARANN

required to localize the faulty components as fast as possible in order to minimize the time for system repairing. It is worth noting that the possible faults may include the physical damages associated with any of the components of the neural network architecture, such as synaptic connections, neuron units and arithmetic cores, as well as memory elements storing weights and activation function lookup tables. Piuri [222] categorized the possible errors into the following classes: unexpected inputs, synaptic errors, summation errors, and errors affecting the non-linear evaluation function within the individual neuron.

Concurrent error detection may be performed by means of traditional data coding techniques, suiting specific neuron implementation [17, 27, 221, 223, 224]: neural operations are performed in the codeword space and, then, results are checked to detect the error occurrence [54, 55]. These data coding techniques give attractive results, without affecting the computational performance in a significant way. Moreover, they do not require any modification of the neural computation, i.e., they do not affect the theoretical characteristics (e.g., learning, recall, and generalization) of the neural network. Localization of faulty neurons is performed by using on-line compact testing techniques [54, 223], through the following two main steps: identification of the layer containing the faulty neuron, and localization of the faulty neuron within the identified layer. On the other hand, some concurrent error detection approaches have been presented with time redundancy. Basically, all of these methods repeat the same computation several times by using the system components in a different way for each repetition. Then the results are compared to detect the presence of errors [116]. One example is the CED scheme [117] in simple arithmetic units (adders and multipliers) using REcomputing with Triplication With Voting (RETWV).

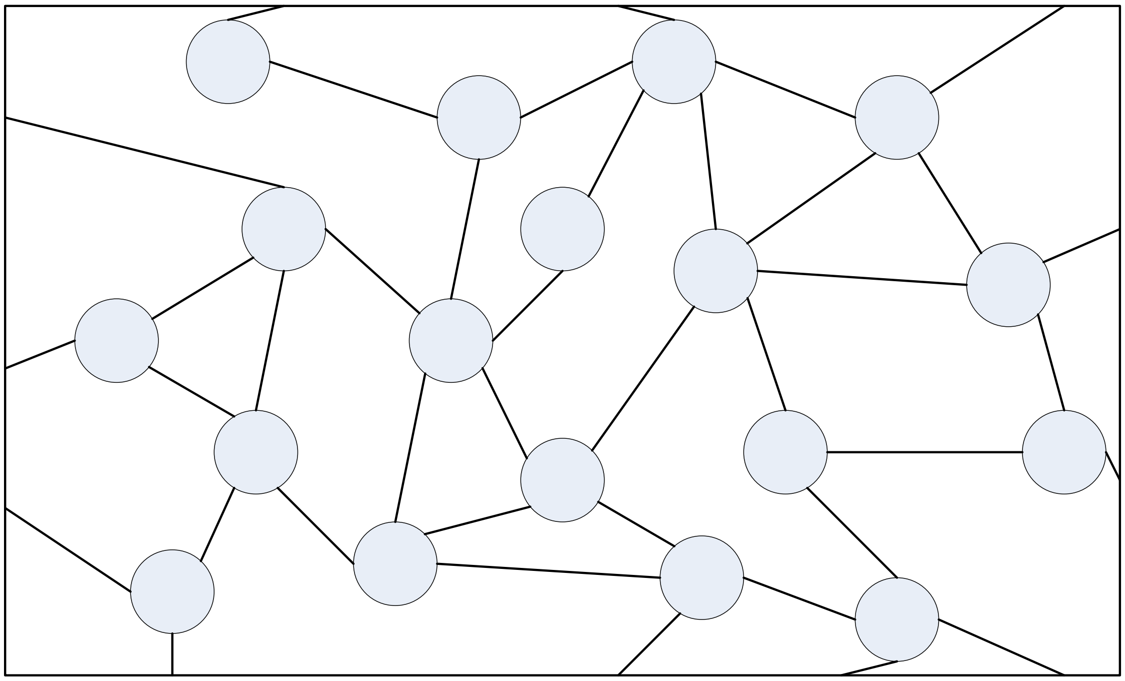
According to the discussion above, it is shown that an appropriate error detection capable of detecting unexpected faults occurring at any neuron and reporting the specific locations of faulty neurons is prerequisite to the subsequent system recovery efforts. Given the locations of faulty components reported by an error detector, as dash-line enclosed in Figure 6, our proposed ARANN architecture can effectively achieve the ANN system recovery by excluding the faulty neurons from the active computation and reconfiguring the network structure in an autonomous manner. In this study, we primarily focus on exploring the fault tolerance and system recovery capabilities of the ANN system from a bio-inspired perspective and thus model the system failure by randomly injecting some faults into active neuron units. Further discussions and implementations regarding more sophisticated error detection approaches that are applicable to neural network systems, have beyond the scope of this research, but which could be considered and evaluated in our future study.

4.2 BIOLOGICALLY INSPIRED APPROACHES

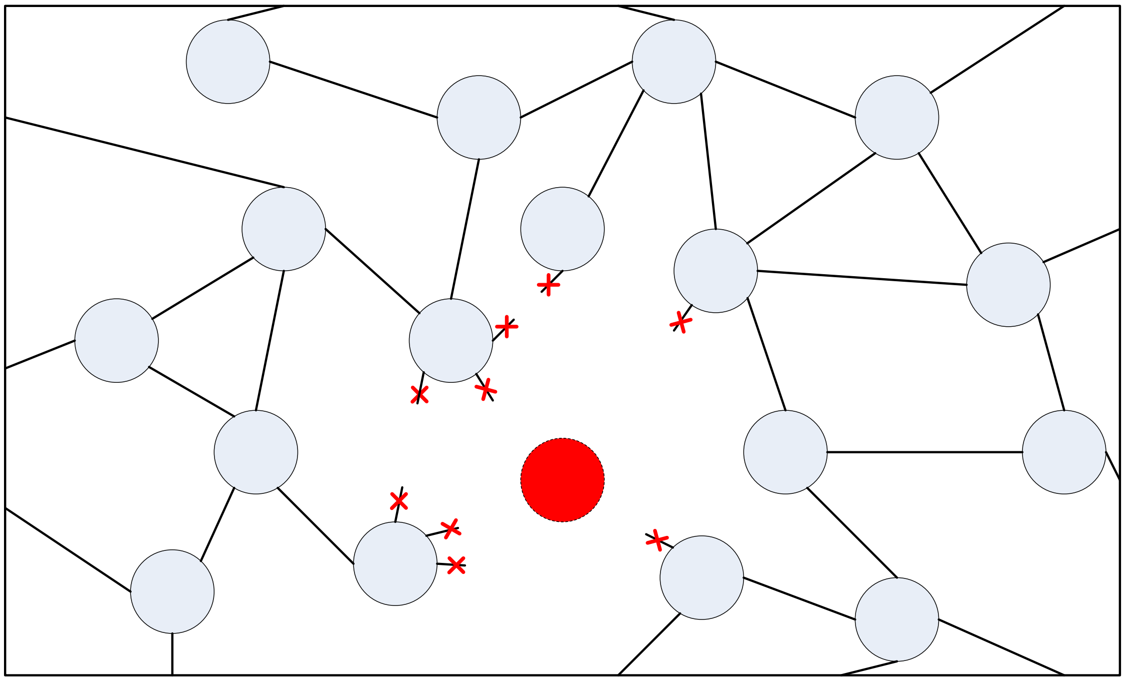
A *neuron* is a biological cell that processes and transmits electrical and chemical signaling through *synapses* [13]. That is, neurons send messages to each other via the synapses, and form a tremendous network of connections, as the core components of the *nervous system*. Within the central nervous system (CNS), the adult human brain usually contains hundreds of billions of neurons and thus is believed to one of the most sophisticated systems in the universe. In an uninjured brain, a huge amount of information is moved, processed, and retrieved within such neural networks to allow us to function normally and support various behavioral actions in our everyday lives. In this Figure 7(a), simplified schematic of a small piece of neural networks has been presented, where many neuron units (gray circles) are interacted via synaptic connections (lines).

When a person suffers a brain injury, caused by either traumatic brain injuries or non-traumatic injuries, many neurons and many more connections between neurons are destroyed, because there are more connections vulnerable to unexpected damage than neurons. In Figure 7(b), a damaged neuron (red circle) is shown in the same neural network piece as Figure 7(a). Accordingly, all connections affiliated with the damaged neuron as well as many other adjacent connections are destroyed. The dead neuron is automatically isolated from the main nervous system, which helps to prevent any potential harmful impacts on the CNS. Now, we would like to examine how our brain further reacts to such injury biologically.

The first mechanism of brain's reaction is referred as *neuroplasticity*, which represents the changing of neurons, network structure and behavioral function, and was firstly identified by William James in 1890 [131]. *Synaptic plasticity*, one form of neuroplasticity, presents the forming process of new synaptic connections between individual neurons. It particularly



(a) A small piece of neural network in the brain



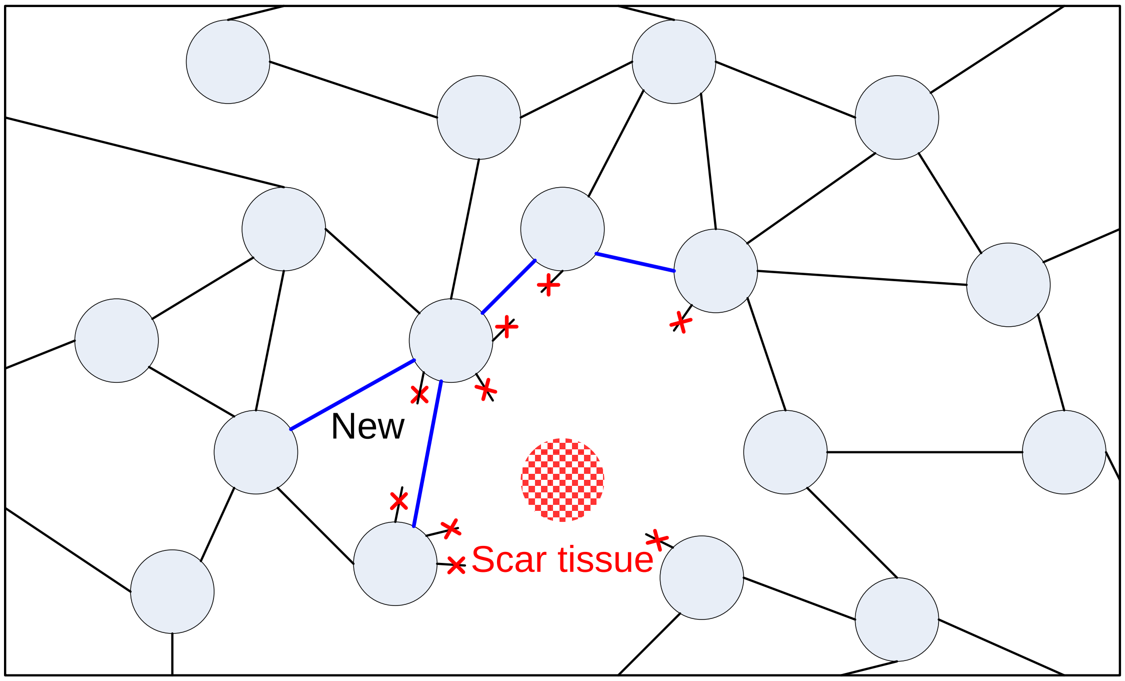
(b) The damage to the neural network caused by a brain injury

Figure 7: Schematic Illustrations of the Damage to Neurons and Synapses

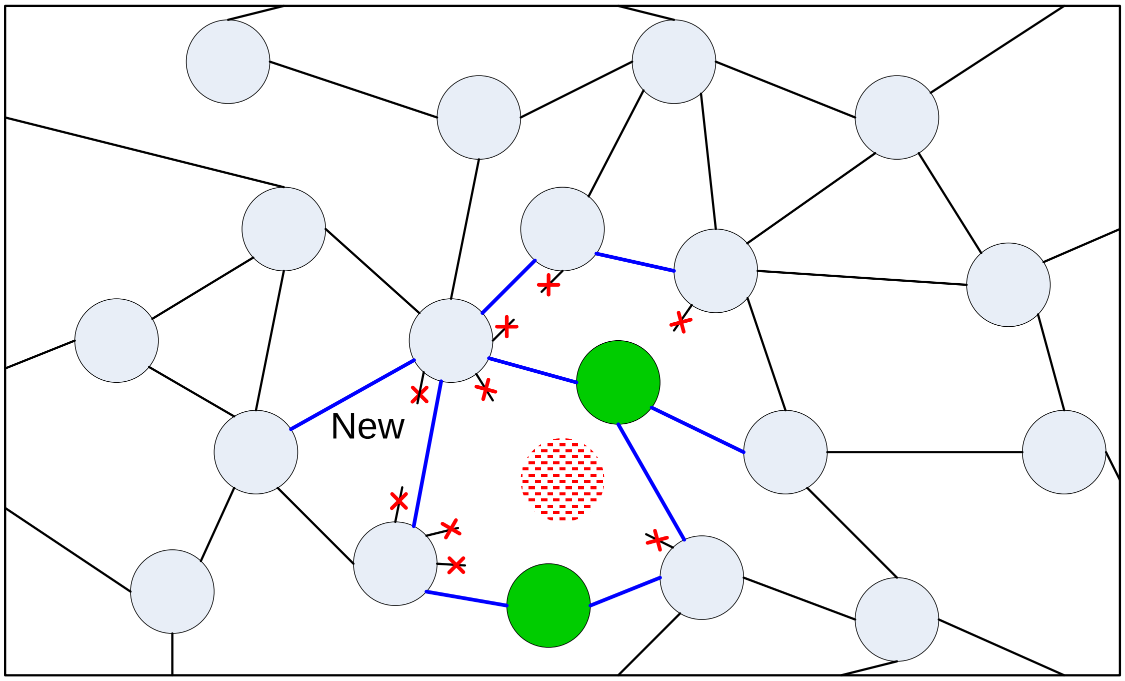
describes the extent to which the brain can reorganize its neural networks in terms of synaptic connections. Given the feature of neuroplasticity, the brain activity associated with a given function can move to a different location, which usually results from the process of recovery from brain injury. That is, in case the neural network is partially damaged, the brain is able to adapt the structure of adjacent areas around damaged neurons and form new synaptic connections to maintain same or at least similar functionality as uninjured status. Figure 8(a) depicts the emergence of new synaptic connections (blue lines) which are formed by the growing and outreaching efforts of new axons to the dendrites of other neurons. As for the dead neurons, the presence of chemicals in brain scar tissue will prohibit the synaptic plasticity [75][272]. Inspired by the principles of neuroplasticity, our proposed ARANN is supposed to be able to automatically remove the damaged neuron units and then adaptively explore alternative network structure and neural connections based on available neurons.

Literally, *neurogenesis* is the process by which new neurons are generated. It was traditionally believed that the central nervous system (CNS) of mammals has very limited regenerative capacity [30]. However, since 1990s, many exploratory research have demonstrated that new neurons are indeed born in restricted regions of the adult mammalian CNS [8][72], in support of Altman& Das's discovery with regard to the continuous neurogenesis throughout adulthood [7]. Recent findings about the addition of new neurons throughout life not only provides a unique model system to understand basic mechanisms of CNS, such as learning and memory, but also raises the promising possibility that stimulation of this process can be applied as a new attractive biological therapy for CNS diseases [186][164].

Given the example of a partially damaged neural network that we used before, Figure 8(b) illustrates the neurogenesis process where new neurons are generated (green units) and new connections to older existing neurons may also be made by these new neurons (blue lines). Inspired by this highly involved biological process, we propose to apply the similar idea onto their silicon counterpart — ANN hardware system. Once a neuron unit in the ANN hardware has been damaged or physically contaminated, besides immediately disconnecting the faulty neuron from the main network, we can also "create" new neuron units and connect them into the neural network to compensate the loss of damaged neuron unit and maintain the "learning" capability of this artificial neural network.



(a) Synaptic Plasticity: reorganization of neural connections



(b) Neurogenesis: Generation of new neurons and new connections

Figure 8: Schematic Illustrations of the Synaptic Plasticity and Neurogenesis

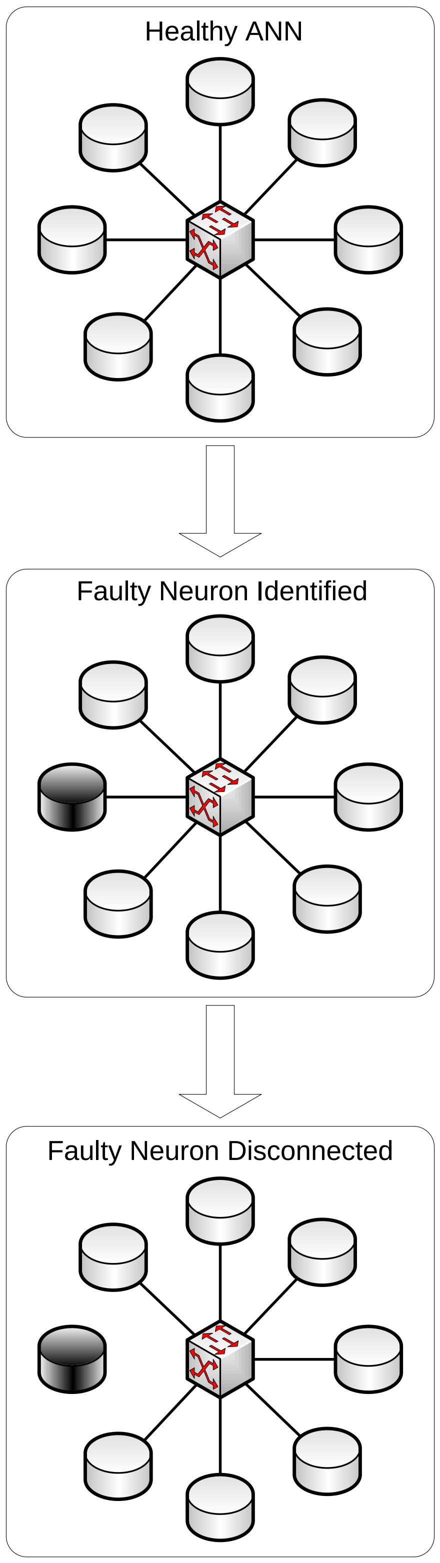
However, in practice, the silicon device itself can not design and generate any new logical modules. One alternative way to achieve the neurogenesis on silicon is to pre-deploy more logic neuron units than needed and activate a few or all of them if certain live neuron units are damaged. Unlike conventional hardware redundancy where redundant components are usually swapped in to replace the exactly identical faulty component, the proposed spare neurons (actually, it may not be appropriate to call them "spare neurons") have the same roles as all other neuron units. They can be directly used in the neural network of a larger size or can be disabled if a small scale network structure is desired. Instead of fairly being the backup alternatives, they are completely designed for any regular use in the neural system. The underlying rationale is to find and incorporate those available neuron modules into the whole neural network to compensate the loss of learning performance caused by damaged neurons as much as possible. Otherwise, all pre-deployed neuron modules can be fully used in a rather large neural network structure to meet more challenging requirements on performance and accuracy. In what follows, we will present the specific architectural innovations to support all aforementioned biologically-inspired features.

4.3 SELF-HEALING ARTIFICIAL NEURAL NETWORK

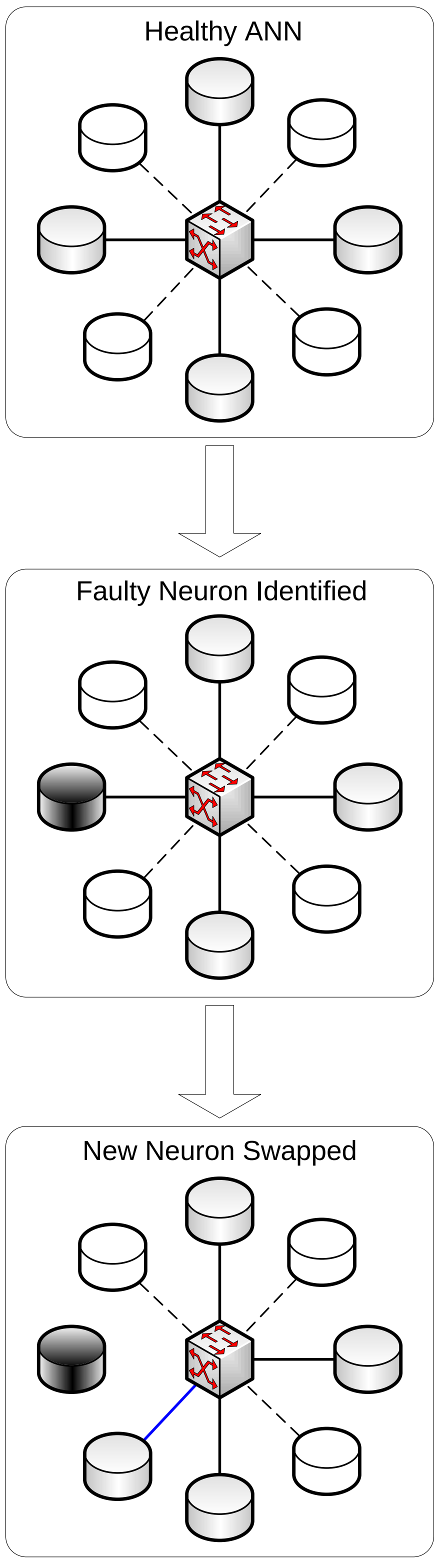
Inspired by the precise, systematic, sophisticated and supremely intelligent automatic recovery mechanism of the mammalian Central Nervous System (CNS) to react to unexpected injuries or diseases, we would like to explore the possibility to mimic CNS's faulty reaction strategies and to develop a cyber-enabled artificial neural network environment with self-healing capabilities. The major motivation of this research is the increasing demands on the computation-effective artificial neural network techniques in biomedicine and healthcare. As we illustrated before, ANNs now have been widely examined for a tremendous amount of biomedical applications and physiological analysis (such as ECG [47, 120, 119, 134, 135, 208, 261], EMG [29, 33, 88, 106, 210, 292], EEG [56, 140, 171, 175, 265, 267], speech [132, 243, 309] and medical image processing [149, 167, 182, 234], etc.) and thus many dedicated ANN-based devices and systems have been developed to facilitate the healthcare and clinical treatment. Particularly, ultra-portable and ultra-reliable ANN-based medical systems have become of great interest because the recent significant development in telemedicine and pervasive healthcare. For the ultra-portable goal, it may not be optimal to design a highly redundant system to provide an exhaustive coverage on any system fault. On the other side, emerging smart biomedical devices will be either worn by people or implanted into human body, thus they are expected to play significant roles in non-stop routine monitoring and directing some medical response mechanisms, such as implanted pacemakers, wearable functional electrical stimulation (FES) devices, or prosthetic limbs. It is easily imagined that how severe problems will be caused, sometimes people would die from this, if the devices can not work properly due to unexpected faults or defects. Meanwhile, it is also unacceptable that the systems have been out of order for a long time and the patients have to wait for some person coming to either repair or replace the broken devices, not to mention that sometimes it would be really difficult to replace a device without invasive surgery.

Therefore, the concept of *autonomously reconfigurability* has emerged as a promising mechanism for ensuring appropriate operational levels during and after unexpected events that could impact or damage the system, especially for those mission-critical systems including the ANN-based biomedical devices. Besides that, the artificial neural network is essentially a highly scalable and parameter non-sensitive architecture, which means the overall system performance is determined by a large set of homogeneous neuron units and their associated synaptic connections, thus the change of a specific synaptic connection or the adding/removing a specific neuron unit will not cause tremendous effects on the whole ANN system. The inherent characteristics of artificial neural networks make the principles of Autonomously Reconfigurability perfectly applicable to the ANN systems and help ANN systems meet the extremely stringent requirements on reliable operations. Given the fact that most previous reliability-aware studies usually enhance the fault tolerance capability of state-of-the-art hardware systems based on either space- or time-redundancy techniques, in this study, we hope to address the reliability issues of emerging ANN-based hardware from another perspective.

The ultimate goal is to achieve a reliable solution and at least maintain appropriate operational capabilities by making ANN system capable of adapting its structure or operations



(a) Automatic Disconnection of Faulty Neurons



(b) System Adaptation with Spare Neurons

Figure 9: Illustration Diagrams of The Neural Plasticity in ARANN

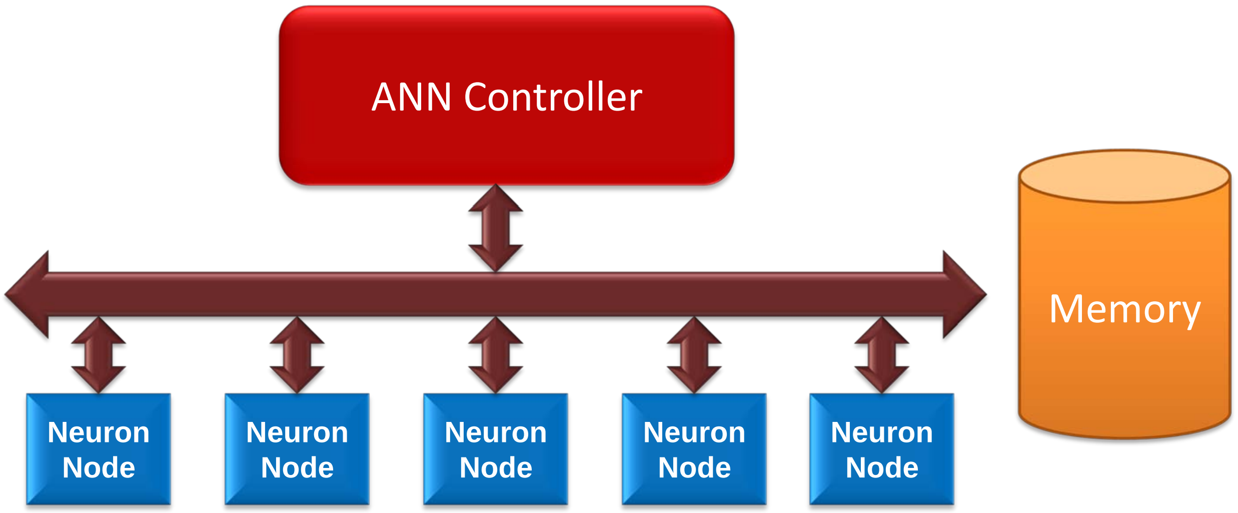
in response to an unforeseen event. This strategic target essentially involves an optimal trade-off among system performance, reliability requirements, and associated costs. Instead of preparing a lot of identical backup hardware components to fill in the vacant positions in case some components are physically damaged, our proposed system would be capable of dynamically determining the ANN system's structure and the connectionism of each individual neuron unit, as well as adaptively finding and incorporating available neuron resources to maintain the best achievable performance of the affected ANN system. Specifically, in a similar way as CNS's recovery process in case of a acquired brain injury, the proposed self-healing ARANN architecture can immediately adapt the system structure to disconnect the damaged neuron unit from the main network, if any error has been reported by the fault detector, and then involve new neuron units into the network to maintain the desired performance if any available neuron units are found (Figure 9(b)). Otherwise, if no further neuron resources are available, the ARANN will continue its normal operation in a compromised mode caused by the slightly fewer neuron nodes contained by the current ANN system (Figure 9(a)). One of the most profound benefits of the proposed ARANN is the opportunity to help ANN system react to any unexpected harmful events in an autonomous, on-line, and efficient manner without halting system execution and introducing considerable redundancy. The details of architectural innovations in ARANN are discussed below.

4.3.1 Distributed Neural Network Architecture

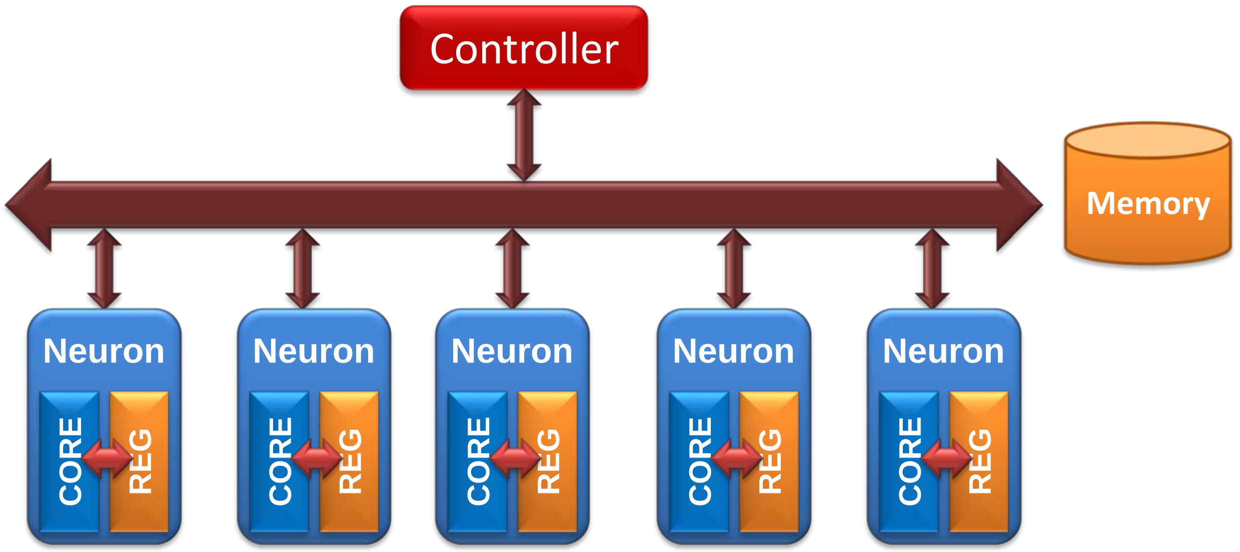
General-purpose computers are traditionally based on the essentially sequential von-Neumann architecture. Inspired by the human nervous system, the artificial neural networks, on the other hand, significantly benefit from their massively parallel processing nature. The inherently parallel nature of ANNs demands a more parallelized computational architecture capable of processing synaptically connected neurons simultaneously using multiple simple processing elements (PEs). Over the past two decades, many research efforts have been initiated to implement and deploy neural networks onto hardware platforms [85, 231, 288].

For the most commonly used multi-layer perceptron neural networks, they are made up of a number of highly connected "neuron" units, which basically are computation nodes capable of performing some fundamental arithmetic operations, and a tremendous amount of “synapses”, which connect neurons with each other and are associated with a “weight” value representing the strength of such synaptic connection. Besides these elements, the neural networks should also have some certain learning mechanisms to direct the training process of ANNs, such as the back-propagation learning algorithm. Intuitively, the hardware implementation of a MLP is supposed to contain similar components to form a neural network, including a number of computation nodes (“neurons”), a memory storing all synaptic weights and results, and a system controller to direct ANN’s operations based on a certain learning algorithm.

In general, neural network hardware designers have followed a more effective and efficient approach, which we called the *Centralized Artificial Neural Network (CANN)* architecture. As shown in Figure 10(a), the CANN architecture has already contained all essential computational components required by an MLP neural network with learning capability. The individual neuron unit is only a computational node capable of performing several types of arithmetic operations. However, such neurons are intended to be simple enough to only process any given input operands according to the designated operation type. The heavy-weight ANN Controller is designed to oversee and manage the whole ANN training or functioning processes, including determining appropriate operations and sequences on each neuron node, allocating and distributing the operand data to each computational neuron node, collecting and processing the results returned from neurons, calculating the total error energy of the ANN to guide the learning process, as well as taking charge of all data communications between neurons and the external memory. It seems that the complicated data processing in the highly structured neural network has been transformed into a semi-parallel computational task, where the ANN Controller will analyze, process, and break up the whole training/functioning computation and then distribute those parallel arithmetic subtasks to computational neuron nodes. The CANN architecture basically tries to convert the highly involved and parallel computations of the neural network into a von-Neumann sequential computing manner, with the benefit of partially parallelized calculations on those independent neuron nodes. Apparently, such CANN architecture is one of the most efficient and simplest implementations of neural networks. It contains a centralized controller and a group



(a) Centralized Artificial Neural Network Architecture



(b) Distributed Artificial Neural Network Architecture

Figure 10: System Architecture Diagrams of the Artificial Neural Network

of rather simple computational nodes ("neurons"), which could greatly alleviate computational burden on the central controller and accelerate the computing progress on the neural network.

However, such CANN architecture also introduces some potential performance overheads. Firstly, the centralized controller is a high potential risk to the reliability of the whole system, because it plays a critical role and has the most complex functions in the neural network system. Secondly, the CANN architecture demands a large amount of data transmission among the controller, memory, and neurons, and thus provides a tremendous bandwidth pressure on the system. For example, at each training stage, the controller may need to access the data from memory, assign them to proper neuron nodes, collect the results from neuron nodes, and then store back to the memory after appropriate processing. It is shown that a massive amount of data-flow has been involved in this process. Thirdly, since the controller exclusively determines all operational sequences, data communications and resource allocations, the scalability of such CANN architecture is very limited. That is, it is unlikely to dynamically modify the system structure and add/remove neuron units to/from the neural network. For different network structures (e.g., different number of neurons), it is mandatory to design and implement a distinct version of the controller. Therefore, it is manifest that such *Centralized Artificial Neural Network* architecture is not applicable for the highly flexible and adaptable neural networks to react to unexpected physical damages.

Given the limitations of CANN architecture, in this study, we propose an alternative approach — *Distributed Artificial Neural Network (DANN)* architecture, as shown in Figure 10(b). Instead of grouping a single centralized controller and a set of basic computational nodes, we propose to design and implement independent, autonomic, smart neuron units, which contains a arithmetic computational core as the neuron node in CANN architecture, a private register file storing all relevant information and intermediate results, and a few control logics directing the operations and data communications of the neuron. Correspondingly, a light-weight ANN Controller is deployed only to direct the ANN training/functioning processes, which means, the current controller will not be involved in any specific arithmetic operations or data communications. The only role of the controller is to instruct each individual neuron unit what they should do now as designated by the learning algorithm, rather than how they do it. On the neuron side, each independent neuron unit now only accepts a series of "instructions" from the controller and then perform corresponding operations, such as memory access, arithmetic calculation, and result writeback. The external memory is also trimmed a lot because all neuron-relevant data now has been stored in their own register files and there is no longer need to transmit a tremendous amount of data between neuron units and the external memory. The memory right now will be purely used for backup and synchronization of ANN state information (i.e., all synaptic weights, generated output results, and magnitudes of weight changes), which will be elaborated in section 4.3.3.

Different from CANN, the proposed DANN architecture has a lot of remarkable advantages. Firstly, the highly independent autonomic neuron units can significantly improve the system performance by maximizing the degree of neuron-level parallelism throughout the ANN training/functioning processes, where all arithmetic operations and data access are carried out simultaneously in each neuron unit. Secondly, the DANN architecture greatly alleviates the burden of data transmissions among controller, external memory, and neurons. Now, there are only the "instruction" and handshaking signals between the controller and each neuron, as well as the ANN state information exchanged between the memory and each neuron. Thirdly, the proposed DANN makes all neuron units behave as a independent processing element, combined with a flexible virtual-to-physical neuron mapping scheme (note: we will discuss it in next section), both of which provide a reconfigurable infrastructure for the ANN structural adaptation and optimization. On the other side, the costs of the proposed DANN architecture include the increased design complexity and resource consumption of neuron units, which may cause noticeably larger chip size comparing with CANN architecture, if a large amount of neuron units are needed.

4.3.2 Decoupled Virtual-to-Physical Neuron Mapping

In last section, we have presented a *Distributed Artificial Neural Network* architecture, which is essentially in favor of highly independent and autonomic neuron units instead of the "master-slave" mode in CANN architecture. With DANN architecture, each physical neuron unit is capable of independently performing any operations as assigned in the neural network algorithm, in a more similar manner as the real human nervous system. In contrast, the ANN controller now only takes charge of directing the system operations and distributing relevant instructions to each neuron. Although the neuron units in DANN have already had much more independent characteristics, their behaviors still rely on the operational instructions provided by the controller. In this case, if one certain neuron unit is damaged, it can not be automatically deactivated unless the controller can be modified manually. Further investigating the functionalities of the controller and neuron units in DANN, they show some important properties of great interest. First of all, on the controller side, it transforms the ANN learning algorithm into a series of instructions to direct the specific operations of each neuron unit. Although the desired number of neurons has been implicitly indicated by the destinations of instructions sent out from the controller, the specific neuron units that will be activated according to the controller instructions are essentially nonsignificant and irrelevant from the controller's perspective. What the controller is really concerned about is the number of neurons that can be used in the neural network as required by the ANN learning algorithm, rather than the presence of one certain neuron. Secondly, on the neuron side, it performs certain operations according to the instructions assigned by the controller. However, since all neuron units are functionally identical, thus they can be used at any location in the whole neural network. In a word, the controller doesn't care about the specific neuron units used in the network work, while the neuron doesn't care about its specific role and location in the neural network. Given this unique characteristics supported by the DANN architecture, we propose a novel *Virtual-to-Physical (V2P) Neuron Mapping* strategy to decouple the "virtual neurons" used in the ANN learning algorithm (i.e., the controller) and the "physical neurons" implemented as individual neuron units on chip, as well as enable ANN systematic adaptation by changing the V2P mapping scheme.

More specifically, as shown in Figure 11, the neurons appearing in the controller are essentially so-called "neuron symbols" or "neuron indexes". That is, any functionally correct neuron units can fill in these positions. Thus, we give a name to these neurons in the controller — "Virtual Neurons". Contrarily, the neuron units physically deployed on chip are named "Physical Neurons". In order to activate and manage some of physical neurons, the instructions that originally assigned to virtual neurons in the controller need to be transferred to the real physical neurons through one possible virtual-to-physical mapping scheme (shown in the middle of Figure 11). This V2P mapping block can flexibly assign the virtual neuron indexes to any physical neuron ports, according to the desired number of neurons and the

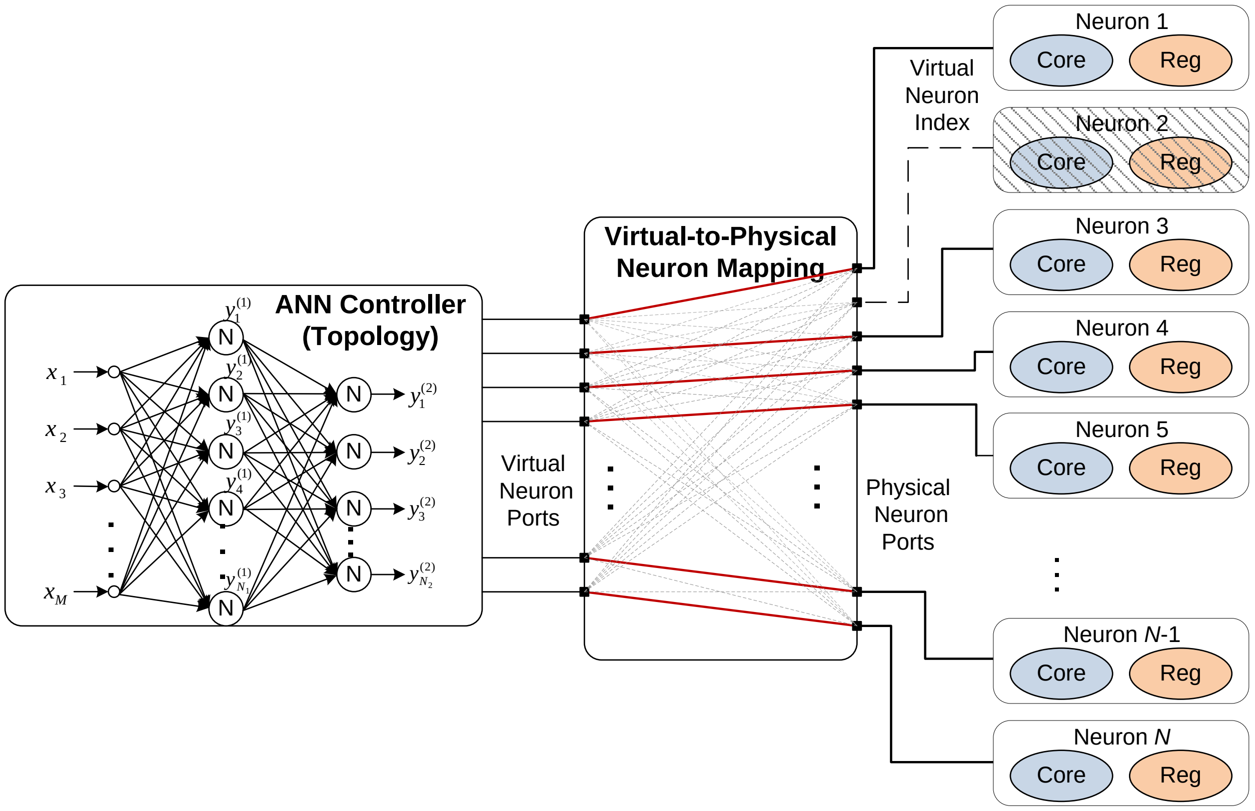
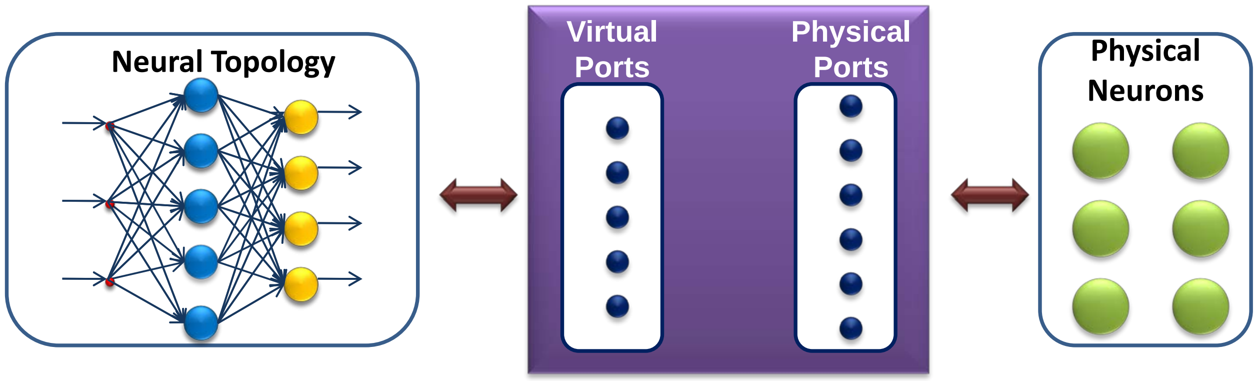


Figure 11: Architectural Diagram of Virtual-to-Physical Neuron Mapping Mechanism

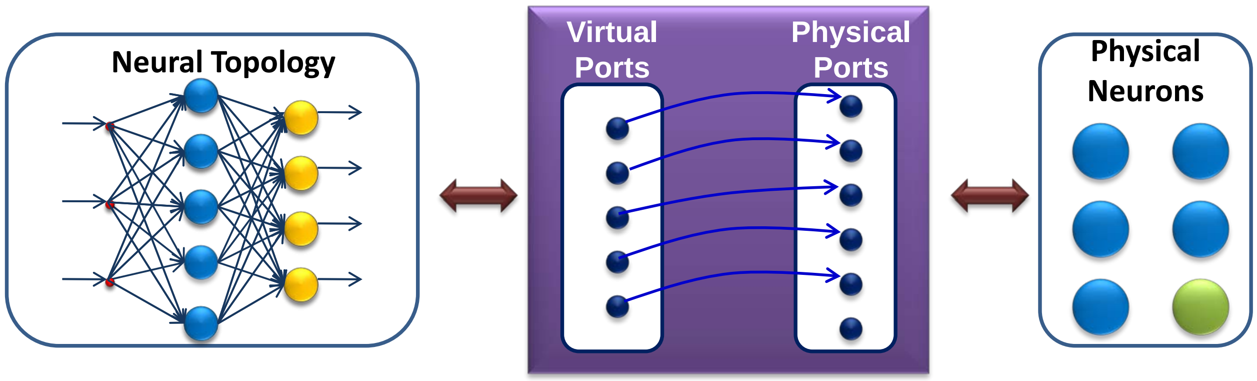
availability of each physical neuron. Once a physical neuron acquires the assignment of a specific virtual neuron index, it will be activated and used in the corresponding location of the neural network as indicated in the training/functioning algorithms.

The proposed *Decoupled Virtual-to-Physical Neuron Mapping* strategy has successfully addressed the reconfigurability and adaptability issues of conventional neural network implementations. It provides a convenient way to achieve the resource-efficient neuron reuse. More importantly, it indicates the possibility of increasing ANN's reliability by automatically reconfiguring and revising its structure in case one or more physical neurons are damaged. In what follows, we will illustrate how the proposed V2P mapping strategy effectively facilitates the neuron reuse and ANN systematic adaptation.

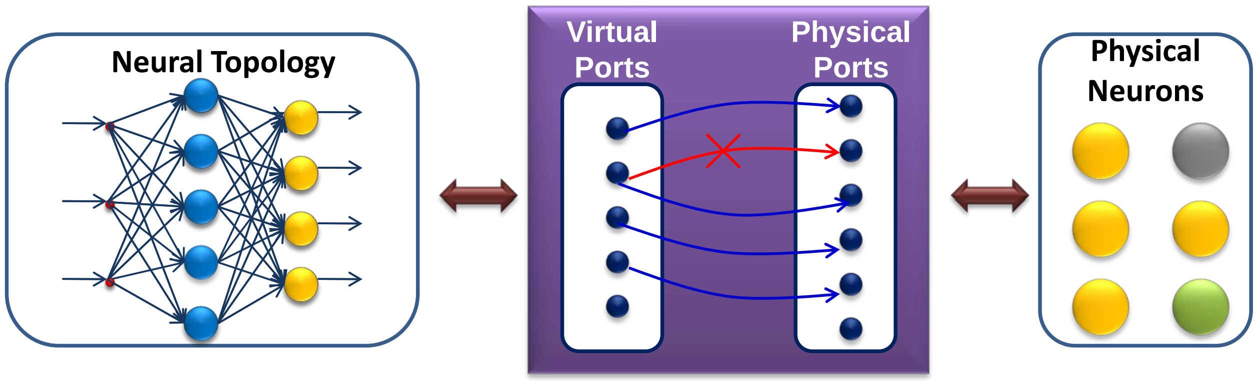
Figure 12(a) illustrates a simple ANN architecture where a Virtual-to-Physical Mapping block has been inserted between the ANN controller and the pool of physical neuron units. According to the neural topology, the neural network structure is made up of 5 neurons in the hidden layer (blue) and 4 neurons in the output layer (yellow). Given the principle of neuron reuse, there are totally six available neurons (green) that have been implemented and deployed. When the controller initiates the instructions to activate the neurons used in the hidden layer, the V2P mapping block will adaptively search the current physical neuron pool and determine an appropriate V2P mapping scheme, as shown in Figure 12(b). Since none of physical neurons is damaged at this moment, the V2P mapping block can easily find a valid mapping scheme, which will accordingly activate the mapped physical neurons and transfer the controller's instructions to these neurons for appropriate operations. When the ANN algorithm moves to the output layer, the controller will update its instructions and dispatched them to fixed virtual neuron ports. Since there is a new update of the instructions and their affiliated virtual neurons, the V2P mapping block will automatically launch a new mapping effort to re-establish appropriate connections between virtual and physical neuron ports. Unfortunately, this time a physical neuron is detected to be faulty, the V2P mapping block will check the availability of each physical neuron, and try to bypass the faulty physical neuron and adaptively search next available physical neuron. Figure 12(c) demonstrates the V2P mapping scheme in case of a faulty physical neuron unit (gray), where four physical neurons (yellow) are activated and used in the output layer.



(a) Decoupled Virtual Neurons and Physical Neurons



(b) Virtual-to-Physical Mapping for MLP Hidden Layer



(c) Virtual-to-Physical Mapping for MLP Output Layer Upon Presence of Damaged Neuron Units

Figure 12: Case Illustrations of Virtual-to-Physical (V2P) Neuron Mapping Mechanism

Figure 12 only shows several simple V2P mapping cases. One other extreme case is that no enough physical neurons available (due to either severe damages or insufficient resources in the physical neuron pool) to meet the structural requirements of the neural network algorithm. In this case, the V2P mapping block can still work properly and use as many physical neurons as possible. Meanwhile, it will return the mapping result back to the controller to indicate a "compromised" mode. This is exactly what we expected for a faulty ANN system: the system either can be completely recovered if there are spare resources available or can be adapted into a "compromised" mode with a certain degree of performance tradeoff. None of these two solutions can be achieved timely and will not stop the system execution. The proposed *Decoupled Virtual-to-Physical Neuron Mapping* strategy is scalable to different scales of neural networks and different availability cases of physical neurons, as well as is also particularly effective for relatively large neural network structure. There are a few possible solutions to implement such a V2P mapping block, we will explicitly elaborate and discuss the specific implementation schemes of the V2P mapping in section 4.5. Given the discusses above, we claim that the proposed *Distributed ANN Architecture* and *Decoupled Virtual-to-Physical Neuron Mapping Strategy* provide a reconfigurable and adaptable architectural infrastructure for the ANN system to react to unexpected faults on the neuron units through dynamic ANN structural adaptations.

4.3.3 Dual-Layer Memory Synchronization Mechanism

Given the proposed *Distributed ANN Architecture* and *Adaptive Virtual-to-Physical Neuron Mapping Strategy* discussed in last two sections, the ARANN now is able to change its structure on the fly, in response to any unexpected error occurring on the physical neuron units. The aforementioned approaches only provide the reconfigurable infrastructure for ARANN and make it possible to heal the partially damaged neural network by removing the defective neuron node and involving new healthy neurons. However, there is another critical issue that needs to be carefully addressed to ensure the ARANN can respond to unexpected hardware errors in a proper manner — the operational consistency of ANN state recovery over time.

It is well known that the standard ANN back-propagation training process (refer to Section 3.2) is made up of a number of training epochs, which contains the three-stage training period (i.e., feed-forward calculation, back-propagation, and weight updating) for each input pattern. The ANN state configurations, including all synaptic weights, predicted outputs and the magnitudes of weight changes, are consistently accessed and updated within each individual three-stage learning period. Also, the ANN training process is highly data dependent, since the magnitudes of synaptic weight changes highly reply on the calculated performance errors, which is iteratively determined by the synaptic weights updated in the training procedure of either previous input pattern or most recent epoch.

Assuming the potential hardware damages occur randomly and they can be successfully identified by the fault detector, accordingly the error information reported by the fault detector can be updated at any time during training. Also we assume that (actually it is mostly the case) the fault detector can detect hardware faults timely but can not all unexpected errors and distribute the newly updated error information immediately after the occurrence of physical damage on neuron units. In other words, when the ANN system receives the updated error information indicating a new fault has been detected on one neuron, the fault has been there for a certain period of time and it may have already destroyed some logic or memory components. Therefore, the recent results generated from the newly identified faulty neuron right before the ANN system receive the updated faulty neuron information may be completely wrong or at least contaminated and suspicious. Unfortunately, those contaminated results may have already been updated to the main memory and synchronized with other neuron nodes. Accordingly, the results recently generated from all other neuron units are all problematic and the whole ANN system has been "infected" even though the faulty neuron has been immediately disconnected and isolated. Considering the severe influence of a faulty neuron on the whole neural network due to the inevitable time delay among the occurrence, detection, notification, and treatment of faulty neurons, a more accurate system recovery scheme besides the systematic reconfiguration is highly demanded to guarantee both the successful recovery of ANN systems in both physical structures and training accuracy. In this section, a *Dual-Layer Memory Synchronization* mechanism is proposed to address the accurate ANN system recovery issue.

As we discussed in Section 4.3.1, the proposed ARANN architecture is essentially a distributed neural network implementation, which primarily include a light-weight ANN Controller, a number of independent smart neuron units, a virtual-to-physical neuron mapping block, and a shared main memory. The ANN Controller is responsible for determining all algorithmic elements and directing the operational procedures of ANN training and functioning. Each physical neuron unit consists of an arithmetic core, a private internal register file, and some other control logics. Given the proposed "smart" neurons, most of highly involved data transmissions and synapse calculations can be finished within each active neuron unit. However, since our proposed ARANN architecture is intended to be resource efficient to meet portability requirements of future wearable/implantable biomedical devices, we propose to reuse all neurons and synapses at different stages of training/functioning (i.e., feed-forward calculation, back-propagation, and feed-forward weight updating) in different layers (i.e., hidden layer or output layer). Although such neuron and synapse reuse can be much more resource efficient comparing with a flat ANN design with dedicated hardware elements for different computation purposes, it also significantly increase the design complexity and raise many challenging issues, particularly the data synchronization problem. For each neuron node, although its private register file has already contained most of intermediate results associated with this neuron, it still requires the results produced from other neurons for the next-stage processing. For instance, $NeuronA$ is firstly used as a output neuron and it generates one output result $Output_A$ as all other output neurons. Then in the back-propagation process, in order to obtain the gradient descent on the squared difference between the desired and actual outputs of the network, $NeuronA$ needs to acquire all other output results generated and stored on other neuron nodes, like $NeuronB$, $NeuronC$ or others. Consequently, a shared memory is needed to backup and synchronize all relevant information from every neuron node involved in the training, in other words, the *ANN State Configurations* including all synaptic weights, outputs of hidden neurons and output neurons, and magnitudes of weight changes.

Figure 13 presents the enhanced *Dual-Layer Memory Synchronization* mechanism. The first layer is a ANN state synchronization at a finer granularity and higher frequency, named *Sync Memory 1*. At each training stage, ANN state configurations from each individual neuron are constantly updated to the *Sync Memory 1* (blue path) and then synchronized with all other neurons (green path). Therefore, the *Sync Memory 1* will keep the most updated information if there is any change on the ANN state configurations. The second layer is basically a ANN state backup at a more coarse granularity and much lower frequency, named *Sync Memory 2*. Once a training epoch is finished, the current ANN state configurations will be immediately backed up to the *Sync Memory 2*, which maintains a more stable record of the latest successful ANN configurations and can be only modified on an epoch-by-epoch basis.

In order to achieve an accurate ANN recovery, we have to design an appropriate system backup mechanism to address the ANN state consistency issue caused by the inevitable time delays among the occurrence, detection, notification, and treatment of faulty neurons. Figure 14 illustrates how the proposed *Dual-Layer Memory Synchronization* mechanism in ARANN architecture can ensure a smooth, accurate, and consistent recovery of neural network systems no matter when a completely unexpected fault is detected.

As shown in Figure 14, assuming an error is detected on a certain neuron unit at the feed-forward weight updating stage during $Input2$ learning in the second epoch ($Epoch2$), unfortunately, it is believed that the just reported error has already occurred for a while before the ARANN's reconfiguration effort to disconnect and isolate the faulty neuron. Thus, many recent results from this neuron are contaminated and suspicious, not to mention that such contaminated results have been propagated to other neuron units making their results problematic as well. Even worse, all these problematic results have been uploaded to the *Sync Memory 1* and polluted the ANN state configurations stored there. In this case, the ARANN will immediately stop the executions in all neuron nodes, initiate the structure reconfiguration through the virtual-to-physical neuron remapping, and direct the ANN Controller to restart the processing flow from the first input pattern $Input1$ as a new epoch. Accordingly, the ANN state configurations stored in the *Sync Memory 2* will be accessed and synchronized with all neuron nodes. Since the error is detected in $Epoch2$ that hasn't finished yet, the ANN state configurations currently stored in the *Sync Memory 2* come from the latest training epoch, i.e., $Epoch1$. ANN Controller's restarting from $Input1$ and the synchronization of ANN state configurations from $Epoch1$ successfully make the system

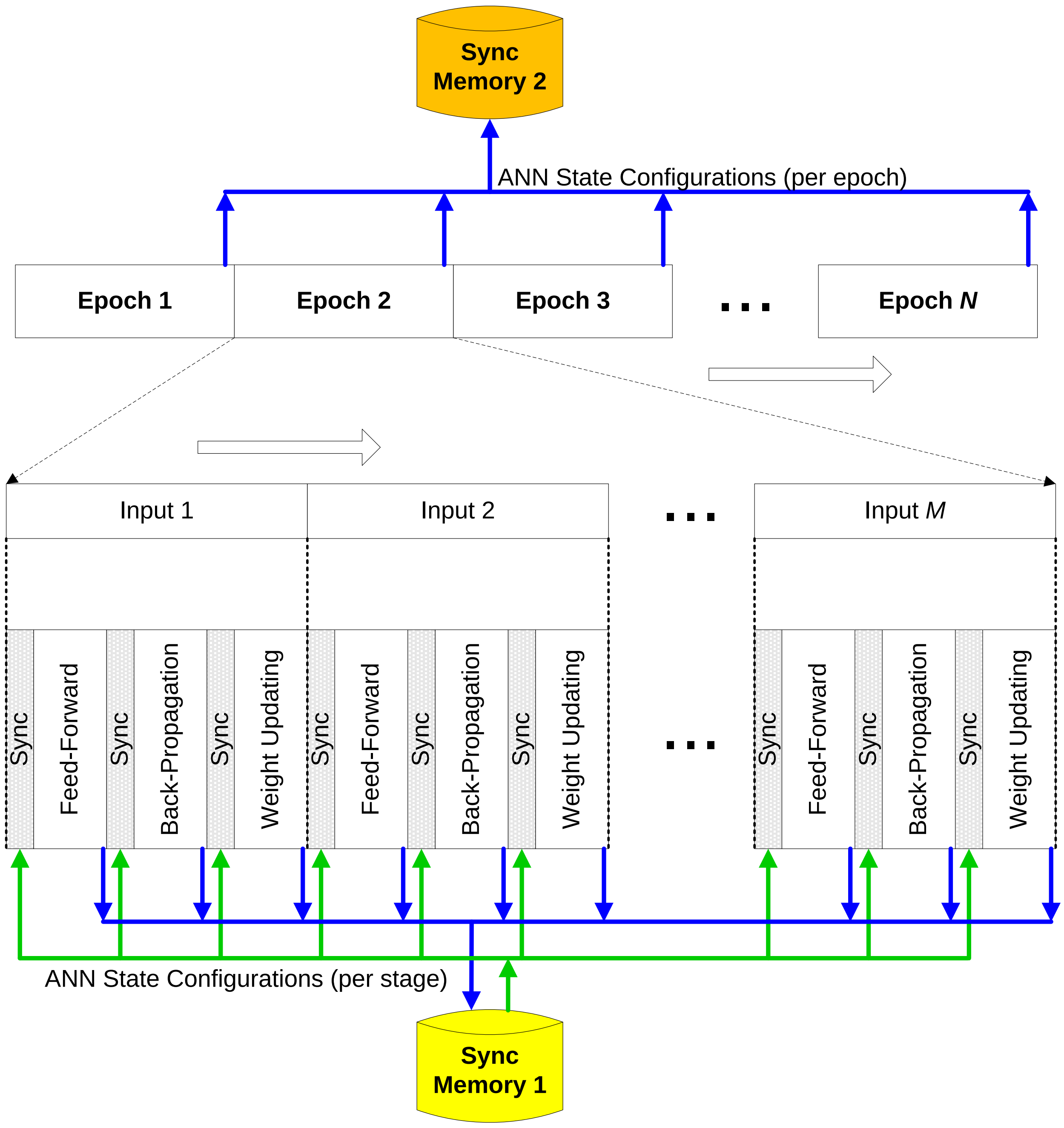


Figure 13: Architectural Diagram of Dual-Layer Memory Synchronization Mechanism

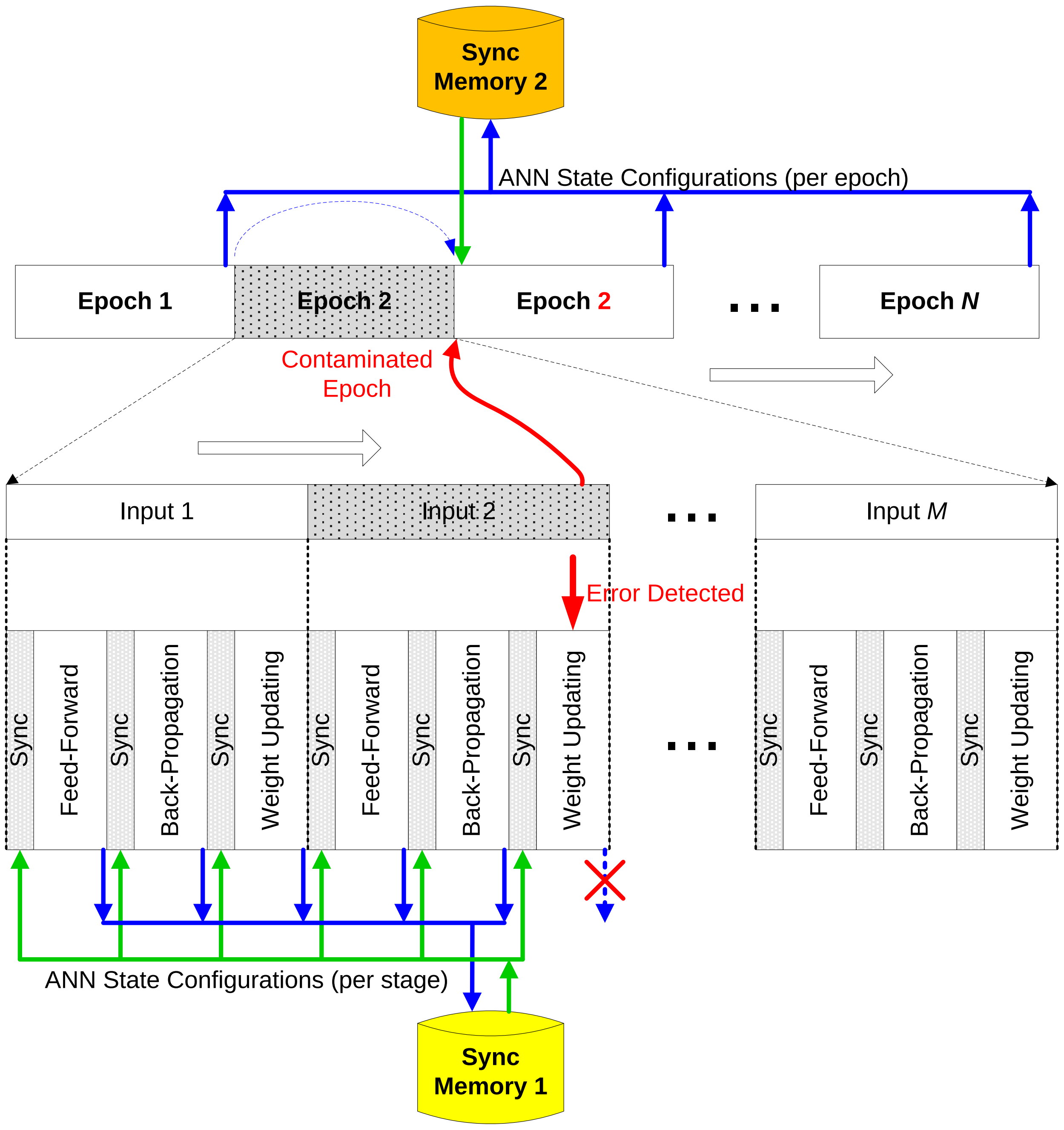


Figure 14: Illustration Diagram of Error Reaction Based on Dual-Layer Memory Synchronization Mechanism

to another new training epoch and bypass the contaminated *Epoch*2. The new epoch has exactly same state configurations information as the original *Epoch*2, except the different physical neuron nodes involved in the neural network.

It is worth mentioning that, the validity and efficacy of the proposed *Dual-Layer Memory Synchronization* mechanism is based on two important assumptions: 1) the time delays among the occurrence, detection, notification, and treatment of faulty neurons are non-negligible, so that the intermediate computation results would be contaminated and invalidated by the faulty neuron before it can be gotten rid of via system structural reconfiguration; 2) any unexpected faults can be detected timely, which means the faulty neuron can be detected and isolated within a reasonable time period after its occurrence, and accordingly it only influence the processing stages within one epoch rather than spreading over several epochs. Based on these two assumptions, combined with effective system structure reconfiguration via V2P mapping, the proposed *Dual-Layer Memory Synchronization* mechanism is able to guarantee the efficient recovery and accurate operation of an ANN system.

What we've discussed until now is for the ANN training mode. As to the ANN classification functioning mode, it has much simpler case since there is no change on the ANN state configurations. The most intuitive and efficient way is just to discard the current input pattern and initiate next input pattern. That is, when a fault is detected, the ARANN will immediately reconfigure its structure and start from next available input patterns, while the invalid results associated with current input pattern are simply discarded. The effectiveness and efficiency of this scheme is claimed particularly based on the fact that most of biomedical applications using ANN are essentially a stochastic process. That means, the specific classification values of quite a few input patterns have limited influence on the clinical diagnostic results unless certain trends have shown up for a relatively long period.

4.4 SELF-OPTIMIZING ARTIFICIAL NEURAL NETWORK

Artificial Neural Networks have been extensively studied and broadly used in a wide variety of applications over the past half century. Along with the remarkable efforts researchers have made to discover more effective ANN algorithms for some as of yet unsolved problems, another important research question of great concern is how to find and determine the best structure and configuration for a given ANN algorithm. Actually, this is a far more efficient way to utilize ANN's powerful computational capabilities and appreciate the considerable benefits of deploying an ANN system. It seems like the computers are still useless if the users do not know how to effectively operate the computers to facilitate their lives, no matter how sophisticated techniques have been built in. As we discussed before, it has been widely agreed that a fully-connected multi-layer perceptron (MLP) neural network provides very satisfied solutions for many real-world problems. For the MLP, a particularly important parameter that affects its performance significantly is the number of neuron units in the hidden layer. Although researchers have proposed many criteria or algorithms to help ANN users explore an optimal structure, unfortunately, there has not been any theory yet to precisely determine the right (optimal) number of hidden neurons used by MLP for a specific problem. Therefore, some other alternative methods, such as so-called *network growing* and *network pruning*, have been proposed to help the users shape the structure of neural networks and remove unnecessary (or "redundant") neurons which have little or no influence on the overall network performance.

Such types of neural optimization strategies have been extensively investigated and used in software implementations of neural networks, however there has not been any neural hardware capable of dynamically optimizing its structure and immediately providing efficient solutions for different applications, as most neural hardware were developed for certain applications only and they are reluctant to evolve into a more efficient shape. However, for emerging wearable biomedical devices and future pervasive healthcare, a highly integrated, multi-functional, ultra low-power, ultra-portable, extraordinary reliable hardware platform is mandatory. As one of the most important and promising techniques, ANN-based hardware is also expected to fit different applications in a more power-efficient manner. Either the strictly fixed network structure or the considerably redundant neuron units, causes severe conflicts with the increasing demands of next-generation biomedical systems on flexibility, versatility, and power-efficiency. One possible solution to achieve this goal is to make ANN adaptable and reconfigurable and thus determine the system structure according to specific requirements and design trade-offs between performance measure and complexity overhead. Given ARANN's unique capabilities of enabling and disabling any physical neuron unit on the fly, it is high desirable to incorporate the structural optimization of neural networks into the ARANN platform. Instead of determining an "optimal" neural network structure for one certain application by the off-line analysis, the ARANN architecture will be able to evaluate the comprehensive system cost involving both performance measure and complexity overhead, and then heuristically explore the most optimal network structure.

4.4.1 Structural Optimization of Artificial Neural Networks

Artificial Neural Networks (ANNs), since its earliest emergence about half a decade ago [104][183][238][289], have been extensively studied and broadly used in a wide variety of applications, such as biomedicine [65][170], industrial control [172][201][269], finance [139][255][307], engineering [43][142], and computer science [89][252]. One of the most attractive features of ANN techniques is a large amount of free parameters that can be adjusted to fit different applications and problems. However, such high degree of flexibility is a double-sided sword, and sometimes may be abused subjectively causing many issues. Usually, for a standard fully-connected multi-layer perceptron neural network, the most critical parameters of great concern are the number of hidden layers and the number of neurons in each hidden layer. It has been widely investigated and demonstrated that, with any of a wide variety of continuous nonlinear activation functions, one hidden layer with an arbitrarily large number of neurons suffices for the "universal approximation" property discussed by Hornik [112][113][114] and Bishop [19] respectively. In this case, the number of neurons in the only hidden layer becomes the only remaining parameter that plays a significant role in determining MLP's behavior and performance. Unfortunately, there has not been any theory yet to precisely determine the right (optimal) number of hidden neurons used by MLP for a specific problem. Although researchers have proposed many criteria or algorithms to help ANN users explore an optimal structure, such as the Akaike's Information Criterion (AIC) [5], Network Information Criterion by Murata et al. [195], and the exploration of best number of hidden neurons [80][173][291], it is still in early stage to widely apply all these algorithms onto real problems due to either their extremely complex algorithmic computations or application-dependent characteristics. Until now, most of previous studies using neural networks have still highly relied on the science of experience or extensive experimental trials. Therefore, a practical issue of using ANNs is how to determine a optimal ANN structure, particularly the number of hidden neurons in the network. In general, the neural network may not learn the presented problem well if it is too small. On the other side, an over-sized network may lead to over-fitting and poor generalization performance [98]. Thus, as we presented before, it is highly desired that the ANN systems can find appropriate network architecture automatically under the guidance of certain algorithms.

When an ANN system is well trained according to given training cases, one important issue that people is usually concerned about is how well it can be generalized to patterns outside provided training set. Actually, the significance of using ANNs is essentially to process or predict new input patterns rather than verify the training data set already provided and used. For a real-world problem usually coming from continuous domains, it is truly impossible to present all possible input patterns to train the neural networks, which also would not be meaningful to do so. If the system pays particular attention to and simply memories the provided training patterns, it may do quite well during the training but fail miserably when presented with similar but slightly different inputs [233]. A highly desirable solution is a well-trained ANN system which can precisely identify the underlying characteristics and inter-node functionality from the training samples as well as can be successfully generalized to any new input patterns providing reasonable answers.

To solve real-world problems using ANNs, it usually requires the use of highly structured networks of a rather large size. A rule of thumb for obtaining good generalization capability is to use the smallest system that will fit the data [233]. Because a neural network with minimum size is less likely to learn the idiosyncrasies or noise in the training data, and may thus generalize better to new data [103]. Since there has not been any theory capable of directly determining the best size of neural networks, we should search and find an optimal network structure by comparing various potential candidates according to a certain evaluation criterion. There are normally two approaches to achieve this, so-called *network growing* and *network pruning*. For the former *Neural Growing* approach, we start with an arbitrary small MLP (a moderate size but not too small to be trained by current input patterns) and then grow additional hidden neurons and weights only when we are unable to meet the design specifications, until a satisfactory solution is found [126]. In contrast, the *Neural Pruning* explore the optimal network structures from another angle. It starts with a rather large MLP with sufficient neuron units for the given application, and train the initial system using a common learning algorithm until an acceptable training accuracy achieved. After that, some inactive neurons will be gradually removed or certain synaptic weights will be eliminated in a selective and orderly fashion. This key idea is to iteratively evaluate the trade-off between the training accuracy and the structural complexity of ANN systems and then select the optimal structure providing reasonable accuracy with the least design complexity. Considering these two distinct optimization strategies, the major drawbacks of neural growing methods are the high risk of getting trapped in local minima and their sensitivity to the initial conditions [98]. When a neural network structure is grown gradually, there is no guarantee that all of newly added hidden neurons are properly activated and trained. Therefore, given the potential limitations of the Neural Growing, in this study we will concentrate on the *Neural Pruning* techniques that starts with a over-sized network structure and then iteratively remove inactive neurons nodes and synaptic connections which are believed to have least contributions to the whole network.

In the recent two decades, many different neural pruning approaches have been proposed and developed. Reed [233] clustered many of the algorithms into two broad groups: one group primarily estimates the sensitivity of the error function to removal of an element; while the other group adds new terms to the objective function representing the complexity overhead that should be taken into consideration. Both sensitivity methods and penalty-term methods can be used for either fine-grained neural structural optimization (i.e., adjustments of synaptic weights) or coarse-grained optimization (removal of inactive neurons). In general, the fine-grained structural optimizations can be integrated into the training process with modified cost objective function and constraints so that weak synaptic connections can be automatically pruned, while the coarse-grained neuron-based optimizations need dedicated evaluation process which includes modifying the trained network, evaluating new cost, and then deciding to either keep the new structure or to retrieve the old one.

Under the category of sensitivity-based methods, Hagiwara [94] presented a simple and effective method for removal of both hidden units and weights, based on the assumption that small weights are irrelevant. However, this is not always the case, especially when small weights are compared to very large weights which may cause saturation in hidden and output nodes. Mozer and Smolensky [194] described a method which estimated the relevance of neuron nodes and then remove the least important ones during training, where the relevance of a unit is defined as the error when the unit is removed minus the error when it is left in place. Karnin [141] measured the sensitivity of the error function with respect to the removal of each synaptic connection and pruned the weights with low sensitivity. Le Cun *et la.* [162] proposed to measure the "saliency" of a weight by estimating the second derivative of the error with respect to the weight. For the penalty-based methods, various different penalty terms have been proposed and incorporated into ANN's cost objective function, which originally only considers the accumulated error energy shown in output nodes. Chauvin [37] added a positive monotonic function into the cost function, which measures the average "energy" caused by hidden neurons. Ji *et al.* [133] demonstrated a method to minimize the number of hidden neurons and the magnitudes of weights by estimating the significance of a hidden neuron with a function of its input and output weights. Weigend *et al.* [293][294] proposed a weight elimination method and added a cost term representing the complexity of the network as a function of the weight magnitudes relative to a specified constant w_0. Ishikawa [125] and Hinton *et al.* [107] respectively proposed two relatively simple penalty terms, called "weight decay", to obtain simplified network structures without complicating the learning algorithm much. Since our research goal is to design an ANN system capable of optimizing itself in a more effective and efficient way for future resource/power-aware biomedical applications, we would like to particularly focus on two most computation-efficient methods: *weight decay* and *weight elimination* in our implementations.

4.4.2 Neural Pruning

When designing a multi-layer perceptron (MLP) neural network, we are essentially establishing a nonlinear model to approximate the complex correlations between the input and output patterns provided in the training data set. We need to carefully manage the training process and determine an appropriate trade-off between the training performance of MLP systems (i.e., accuracy of predicted outputs) and the potential capability of being generalized to other data patterns. In the context of back-propagation learning, we may realize this trade-off by minimizing the total cost associated with the current network structure [103]:

$$R(\mathbf{w}) = \mathcal{E}_s(\mathbf{w}) + \lambda \mathcal{E}_c(\mathbf{w}) \tag{4.1}$$

The first term, $\mathcal{E}_s(\mathbf{w})$, is the standard *performance measure*, which is determined by both neural network structure and current input patterns. In back-propagation learning, it is typically defined as a accumulated mean-square error of all output neurons and will be evaluated for each training input/output pair on an epoch-by-epoch basis. The second term, $\mathcal{E}_c(\mathbf{w})$, is the *complexity penalty*, which only depends on the network structure and will reward ANN systems with lower complexity. Besides that, the λ can be regarded as a *regularization parameter* representing the relative importance of the *complexity penalty* term with regard to the *performance measure* term. Considering two extreme cases: 1) when the λ is zero, ANN's training process is completely driven by the pursuit of maximum performance (minimum error energy); 2) when the λ is infinitely large, in contrast, now the training of neural networks will be stringently constrained and determined by the desired system complexity.

As we discussed above, the two most effective and efficient complexity penalty functions are the ones defined in *weight decay* and *weight elimination* approaches. It is worth mentioning that either *weight decay* or *weight elimination* is not the strictly correct form of complexity regularization for a multi-layer perceptron, according to Haykin's analysis [103]. However, due to the speed/power/resource requirements of ARANN's potential applications, we are in favor of a rather efficient optimization method capable of helping ANN system find the most optimal structure automatically. Actually, it is widely agreed that these two forms of complexity penalty are simple and work well in many cases.

In *weight decay* procedure, the complexity term is defined as the squared norm of the weight vector $\mathbf{w}$ (i.e., all available synaptic weights and biases) in the network [107]:

$$\mathcal{E}_c(\mathbf{w}) = \|\mathbf{w}\|^2 = \sum_{i \in \mathcal{E}_{\text{total}}} w_i^2 \tag{4.2}$$

where $\mathcal{E}_{\text{total}}$ refers to all the synaptic weights in the network. The integration of weight decay complexity penalty term into overall cost function will help the MLP network trim some synaptic connections that have little or negligible influence on the network.

Similarly, Weigend *et al.* proposed a more complicated complexity penalty term representing the complexity of the network as a function of the weight magnitudes relative to a specified constant [293][294]:

$$\mathcal{E}_c(\mathbf{w}) = \sum_{i \in \mathcal{E}_{\text{total}}} \frac{(w_i/w_0)^2}{1 + (w_i/w_0)^2} \tag{4.3}$$

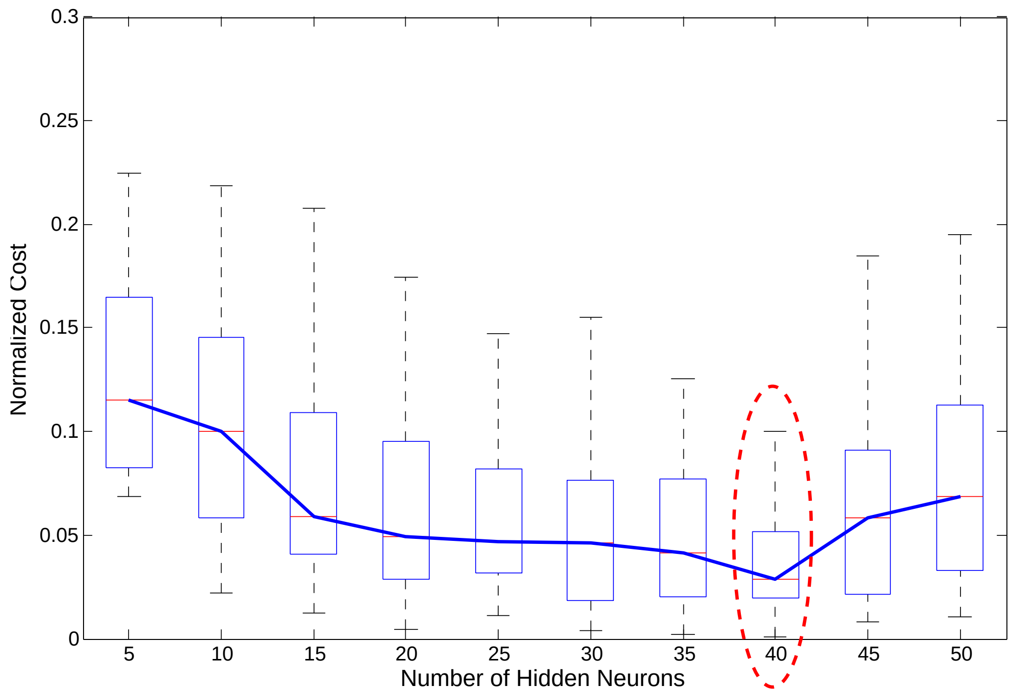
where w_0 is a preassigned parameter, and w_i refers to the weight associated with the synaptic connection i in the network. When $|w_i| << w_0$, the complexity penalty is approaching zero and it is indicated that this synaptic connection should be eliminated from the network. On the other hand, when $|w_i| >> w_0$, the complexity penalty is approaching the value of one, which indicates a significant synaptic connection within the network.

In order to verify and demonstrate the effects of complexity penalty terms on the selection of appropriate number of hidden neuron units, we conducted an experiment using the data from our ANN-based locomotion prediction case study (section 3.3). As introduced before, in this case we have totally 127 clinical trials from 12 individual subjects. 14 electromyograph (EMG) signals and EMG co-activation variables are recognized as the inputs to the ANN model. To clearly present the change of cost function values purely caused by the complexity penalty terms, here we only focus on one single locomotion variable — stride length, which will be the only output of the ANN model. To further increase the generality of our discussion, we randomly constructed 50 training data sets (i.e., 50 validation data sets correspondingly) out of the overall 127 data examples according to the specified selection rules (i.e., 5/6 of 127 examples are used as training data and the others are used as validation data.). Given the 50 randomly selected training data sets, we performed 50 independent network training for each network structure, which varies with different number of hidden neuron units spanning from 5 to 50. The final normalized costs associated with all of trained network structures are collected and presented in Figure 15.

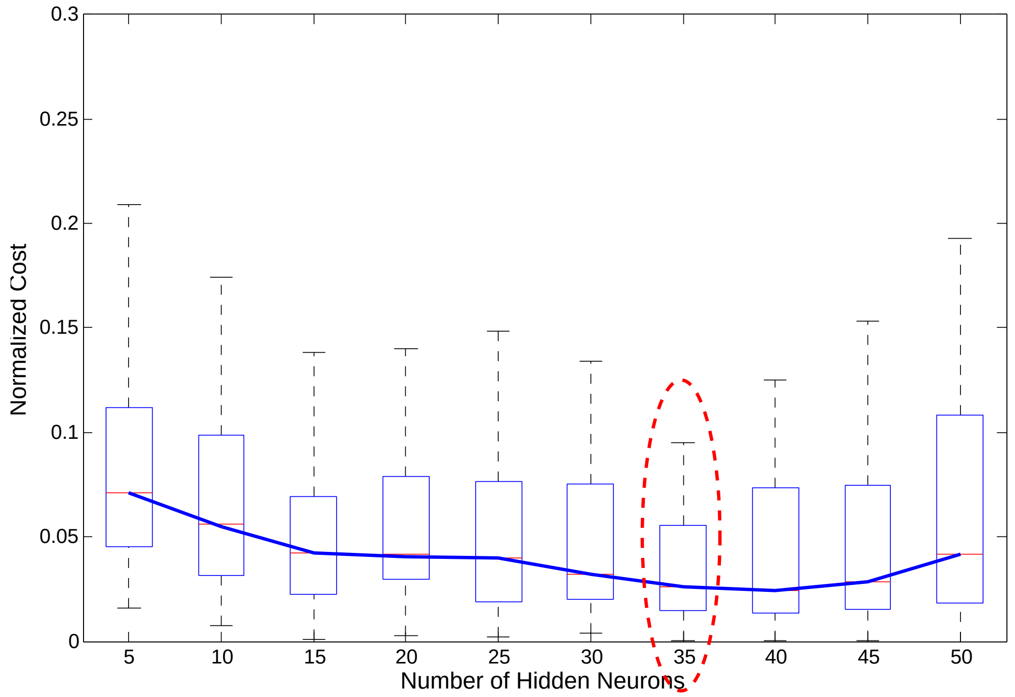
Figure 15(a) shows the distribution of normalized costs of 50 independent training processes over 10 distinct network structures. We can observe that, if only the conventional performance measure (i.e., error energy) is considered, the ANN model will have the best training performance and the least cost when it contains 40 hidden neurons. When the complexity penalty term defined in Equation 4.2 is added into the cost function, the cost distribution of 50 training processes changes accordingly, as shown in Figure 15(b). In this case, rewarded by less neuron number and less synaptic connections involved, the neural network with 35 hidden neurons shows the least cost and achieves an appropriate trade-off between training performance and network complexity. The *regularization parameter* λ used in this experiment is assumed to be one, which means the training performance and network complexity are believed to be equally important to the model decision. The experiment will show slightly different results if a certain preference is given to either training performance or complexity overhead. Also, it is agreed that simpler network structure with rather reasonable training accuracy will be in high favor if the factor of complexity is considered, no matter what type of complexity penalty terms are defined (e.g., the penalty terms defined in Equation 4.2 and 4.3).

4.4.3 ARANN-based Self Optimization

Although such type of optimization strategies has been extensively investigated and used in software implementations of neural networks, there has not been any neural hardware capable of dynamically optimizing its structure and providing efficient solutions for different applications, because most neural hardware were developed for certain applications only and they are reluctant to evolve into a more efficient shape. However, for emerging wearable biomedical devices and future pervasive healthcare, a highly integrated, multi-functional, ultra low-power, ultra-portable, extraordinary reliable hardware platform is mandatory. As one of the most important and promising techniques, ANN-based hardware is also expected



(a) Cost Function without Complexity Penalty Term



(b) Cost Function with Complexity Penalty Term (as defined in Equation 4.2)

Figure 15: The Normalized Cost over Different Numbers of Hidden Neurons

to fit different applications in a more power-efficient manner. One possible solution to achieve this goal is to make ANN adaptable and reconfigurable and thus determine the system structure according to specific requirements and design trade-offs between performance measure and complexity overhead. Given ARANN's incomparable capabilities of connecting and disconnecting any physical neuron unit to/from the main network on the fly, it is high desirable to incorporate the structural optimization of neural networks into the ARANN platform. Instead of determining an "optimal" neural network structure for one certain application by the off-line analysis, the ARANN architecture will be able to evaluate the system cost involving both performance measure and complexity overhead, and then adaptively explore the most optimal network structure with the appropriate design tradeoff in a way similar to neural pruning.

Figure 16 presents the workflow of the proposed adaptive self-optimizing ARANN architecture. Basically, it add extra evaluation modules to assess the system cost according to certain form of cost functions. Since the ANN hardware is not able to emulate various network structures at the same time, we can not evaluate a number of trained networks simultaneously and simply select the best one out of all candidate structures. Therefore, inspired by the neural pruning techniques, we propose to implement a simplified adaptive optimization technique to find an optimal network structure dynamically. The detailed steps are illustrated below:

1. Start from the default network structure, which uses all available physical neuron units in the network or an estimated number of neurons believed to be large enough for the current application.
2. Following the standard back-propagation training procedures, the weights associated with synaptic connections between neurons will be updated. The updated network configurations (i.e., all weights and biases) will be backed up to the *Topology Memory I* for each input pattern, thus the Topology Memory I will keep the most recent neural structure configuration.
3. Once all input patterns have been presented to the neural network, one training epoch is finished and the accumulated error energy of such epoch is recorded. This process will be repeated until the training termination criterion has been met, i.e., the change

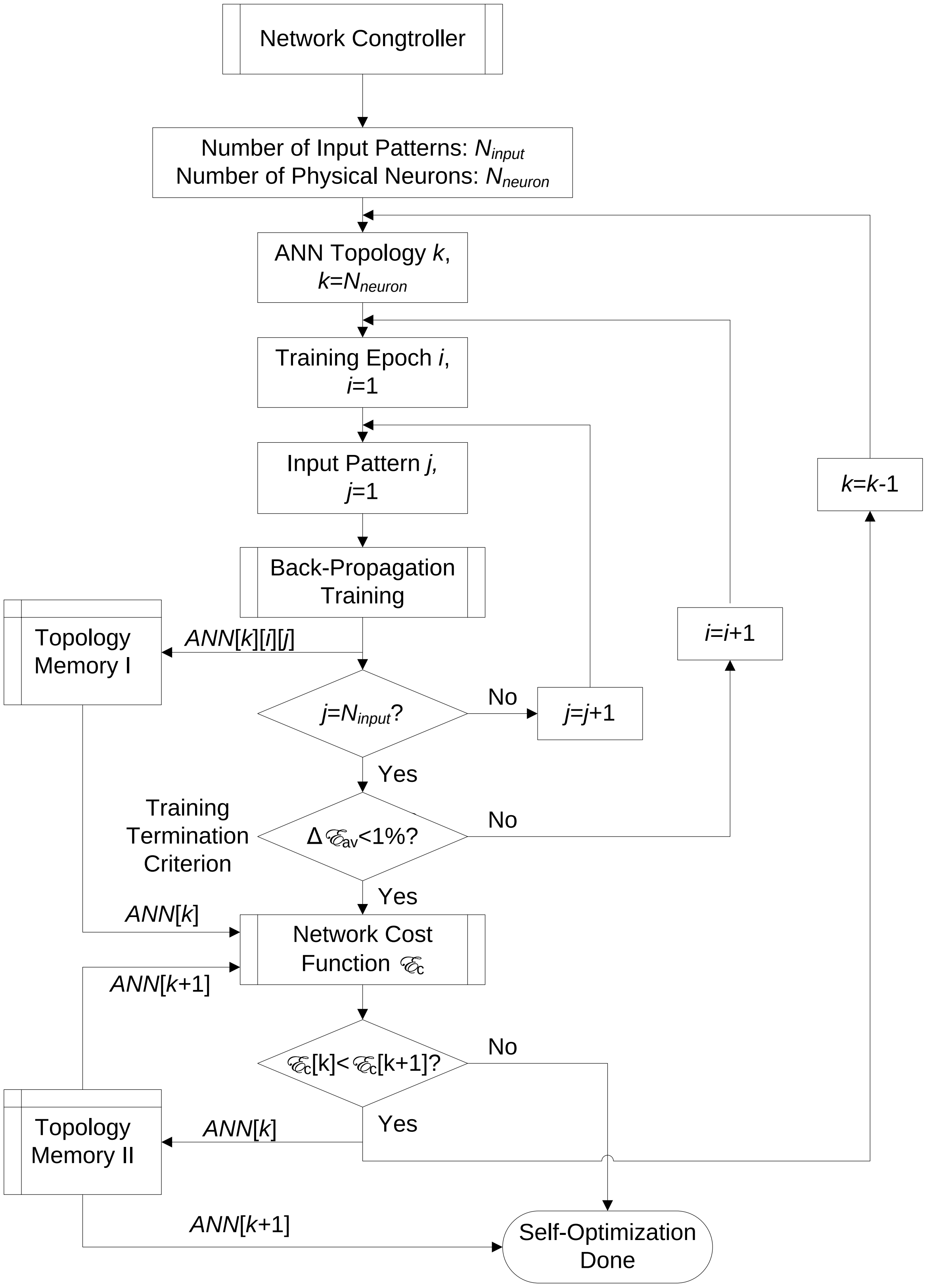


Figure 16: Adaptive Self-Optimization Workflow of ARANN

rate of error energy over each epoch is small enough (Note: we will discuss the training termination criterion below).

4. When the termination criterion is met, it is believed that the training process has already converged into a reasonable stable status and the neural network has been well trained. The current network configurations will be used to calculate the system cost according to the given cost function, including both performance measure term and complexity penalty term.
5. Compare the cost value $\mathcal{E}_c(\mathbf{w})[k]$ of current trained network with k neurons with the cost value $\mathcal{E}_c(\mathbf{w})[k+1]$ of the last trained network with $k+1$ neurons. If $\mathcal{E}_c(\mathbf{w})[k] < \mathcal{E}_c(\mathbf{w})[k+1]$, it is shown that the current network configuration has a better overall performance than the previous one, which use one more neuron in the network. Correspondingly, the current network configurations will be backed up to the *Topology Memory II*, which maintains all information of the latest trained network, denoted as $ANN[k]$. Otherwise, if $\mathcal{E}_c(\mathbf{w})[k] > \mathcal{E}_c(\mathbf{w})[k+1]$, it represents that the system has started to show increasing trend on the system cost and further pruning neurons may hurt the overall performance of neural networks. In this case, the network with $K+1$ neurons is believed to provide the optimal trade-off between performance measure and complexity overhead. Therefore, the network configurations will be retrieved from the Topology Memory II, which always maintains the configuration information of the latest neural network structure that has been assumed to have the best performance until now.

Following the steps described above, the ARANN starts from the default structure with all available neurons and then heuristically searches the most optimal network structure by disconnecting one neuron unit at a time. The proposed double-backup-memory architecture helps ARANN differentiate the most recent network configuration and the latest optimal one, as well as maintain them in two distinct threads. Once a larger cost value is recognized, compared with the cost associated with the latest optimal network structure, the system will automatically access the Topology Memory II and recover the whole neural network using stored configuration information. There is no need to re-train the network again and an optimal neural network will be put into use immediately. We will demonstrate the efficacy and efficiency of the proposed Self-Optimizing ARANN in the section 6. On the other side, however, it is worth mentioning that, although such a neural pruning-like heuristic optimization process will effectively guide the network towards the most optimal structure with appropriate design trade-offs, there is a possible risk that the neural system will get trapped into local minima since ARANN does not evaluate all possible structures simultaneously and a local minima may prevent ARANN's further exploration of a better network structure.

As to the training termination criterion, since the back-propagation algorithm cannot always show a clear convergence trend in most practical applications, thus there are no well-defined criteria for stopping its operation. As a reasonable alternative, Haykin [103] presents an efficient termination criterion that can approximately capture the convergence trend of back-propagation training process and thus stop the network parameter adjustments timely:

"*The back-propagation algorithm is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small.*"

It is typically considered to be small enough to terminate the training process for the rate of change in the average squared error $\mathcal{E}_{av}(\mathbf{w})$ if it is roughly equal to or less than 1 percent per epoch. Therefore, in our following optimizing experiments, we all use the 1% change rate as the training termination criterion for each neural structure setting. Once a training process is stopped according to the given termination criterion, it is assumed that the current MLP structure has been well trained and the accumulated error energy in output ports is able to reasonably represent the performance of the current network structure. Then, incorporated with the complexity penalty functions discussed in section 4.4.2, we can iteratively evaluate each network structure with decreased number of hidden units and adaptively find the most optimal MLP neural network structure with appropriate trade-off between the training accuracy, generalization capability, and design complexity.

4.4.4 Power/Thermal-Aware Design Optimization

As electronic circuits' speeds and circuit densities continuously increase, circuit board power density increases as well and thermal management becomes an increasingly significant part of system design [25]. During the development of a large-scale circuit board, thus the thermal design aspects have proved crucial to its reliable operation. Reducing thermally induced stress and preventing local overheating remain major concerns when optimizing the capabilities of modern system chips [24]. However, such thermal-efficient approaches will usually bring considerable loss of performance, which is also critical to the increasingly computation-intensive applications. Therefore, seeking an effective way to balance the requirements on the high computational performance and the reliable operations with efficient power management has been of great interest to the academia. The employment of reconfigurability design concept can bring extra benefits in further addressing the reliability issues during the system execution. In this section, we will discuss how the system's reliable operation can be guaranteed and the overheating issues can be avoided by integrating the autonomous reconfigurability into our conventional ANN platform.

An intuitive way to prevent the system from overheating without loss of performance is to provide more modular design options that can be conveniently loaded and integrated into the main system. These design options may offer different performance/power tradeoffs and many other controllable diversified characteristics. When the system is started, the default configuration consists of all modular performance-optimized components in order to achieve the desired performance requirements. However, the performance-optimized designs usually need more power consumption and correspondingly cause potential overheating problems after the system has continued execution for a certain long time. In this situation, another design component with the exactly same functionality but designed targeting power efficiency can be loaded to replace the original high-performance design modules. With the support of the proposed Virtual-to-Physical Neuron Mapping, such swaps between characteristics-specific modules can be accomplished within a little while. Once the system has been cooled down by switching to power-efficient design modules, the high-performance system components can be now reloaded and re-integrated into the system again. In this way, the complex system can achieve an optimal balanced tradeoff between the intensive performance demands and the robust reliability requirements.

In this study, we would like to show a simple example on this reconfiguration-based reliability augment methodology. The artificial neuron nodes, representing the massively complicated computational operations performed within the human being's nervous system, are believed to be the most computation-intensive modules. Furthermore, since a large number of identical neurons are placed onto the FPGA, various design strategies applied on a single neuron design can be further augmented and multiplied when deploying all neuron nodes in the network.

We designed and implemented two versions of neuron nodes, targeting for the performance-intensive requirement and power-efficient demand respectively. Their specific design characteristics are shown in Table 4. It is demonstrated that the high-speed neuron can achieve more than 80% speedup than the power-efficient neuron. Correspondingly, the power-efficient neuron is able to save design logics by 50% and power consumptions by around 10%. This two simple design options provide more flexibility to adapt a robust, reliable system and achieve an optimal balance between the system performance and power consumption.

4.5 VIRTUAL-TO-PHYSICAL NEURON MAPPING

As we presented before, the Virtual-to-Physical (V2P) Neuron Mapper is one of the most critical components within this Autonomously Reconfigurable Artificial Neural Network (ARANN) architecture and also the major element which introduces extra time and space overhead to the ANN system. In this section, we explore several different V2P mapping implementation schemes and analyze their specific performance characteristics and applicabilities to pursue the lowest time and space overhead associated with autonomous reconfiguration capability. Given the desired number of neuron units (determined by the ANN Controller) and the locations of potentially damaged neurons (designated by the Error Detector), the V2P Mapper will establish connections between the virtual neuron ports and corresponding physical neuron units. There are generally two cases associated with such V2P mapping process. The first case is that the available (physical) neuron units in hardware are more than the desired (virtual) neurons specified by the ANN Controller, thus like those faulty neurons, some neuron units will not be enabled and used in the current ANN structure. The other case is that the available physical neuron units are not enough to meet the needs of the ANN Controller any more, probably due to gradually increased damage on

Table 4: Comparison of Reconfigurable Neuron Implementations

Properties		**High-Speed Neuron**	**Area-Efficient Neuron**
Area[a]	Flip-Flops	296	190
	Slice LUTs	559	367
	Route-Thrus	3	34
	Occupied Slices	219	114
	bonded IOBs	61	61
	DSP48E	1	3
	Total Equi. Gates	6750	4539
Performance[b]	Delay	3.599ns (L:0.965ns/R:2.634ns)	6.587ns (L:5.179ns/R:1.407ns)
	Offset Before 'CLK'	3.216ns (L:0.831ns/R:2.385ns)	3.937ns (L:1.195ns/R:2.742ns)
	Offset After 'CLK'	2.502ns (L:2.285ns/R:0.217ns)	2.502ns (L:2.285ns/R:0.217ns)
	Frequency	277.855MHz	151.814MHz
Power[c]	Total Est. Power	506mW	473mW

[a] The device utilization data was obtained based on Xilinx Virtex-5 XC5VLX50T FPGA.
[b] The timing information given within the parentheses refers to Logic – (L) and Route – (R) respectively.
[c] The power consumption data was estimated primarily based on static behaviors by XPower tool.

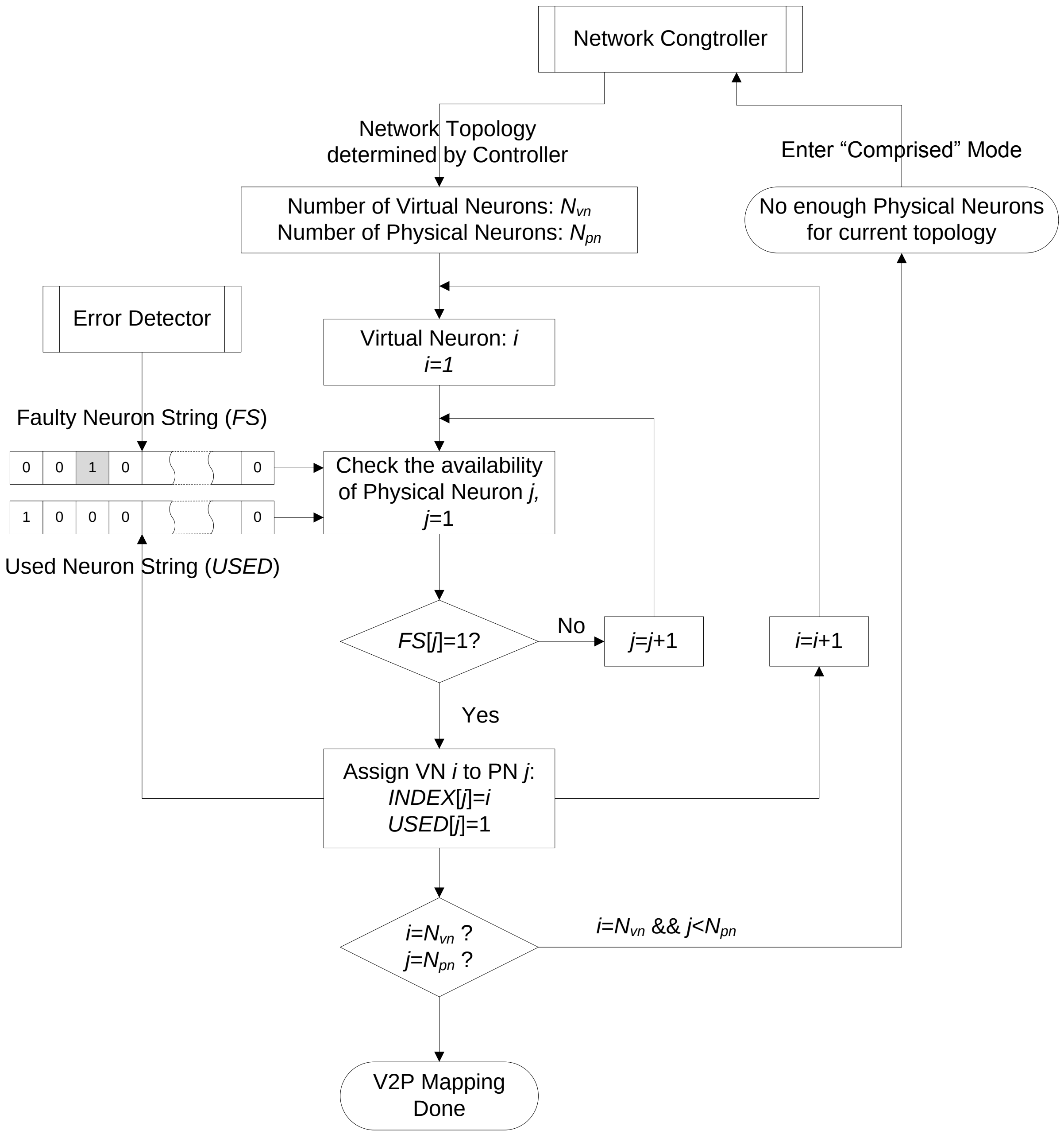
hardware. In this case, the V2P Mapper will exhaustively search those still "healthy" neuron units and involve all of them in the current ANN structure. Also, the V2P Mapper will return the number of currently involved physical neuron units and a feedback signal back to the ANN Controller to tell users that the system is now running in a "Compromised" mode and the level of damage on hardware.

Considering the possibility that the electronic reliability issues will become increasingly severe and the exponentially growing needs of more versatile, easily configured ANN hardware, it is highly desired to design and implement a fast, flexible, accurate, and resource-efficient V2P mapping block which can be integrated into our ARANN architecture. In this section, we will present several different V2P mapping design solutions from various perspectives and then compare their performance, implementation efficiency, and potential overhead. It is worth mentioning that the appropriate selection decision of an optimal implementation strategy highly replies on the specific design considerations and system applicability, such as the expected level of fault/defect occurrence, the estimated frequency of ANN system adaptations, and the speed-area-power trade-off. Our findings will show some basic profiles of each design choice and provide general guidelines for an optimal design solution.

4.5.1 Adaptive Physical Neuron Allocation

Intuitively, the first strategy is to design a dedicated V2P mapping logic block. As we explained above, this V2P mapping block accepts a faulty neuron bit string provided by the Error Detector, which represents the specific locations of damaged neurons, and the desired number of virtual neurons specified by the ANN Controller. After an exhaustive search, such V2P mapper will be able to determine the appropriate connections between virtual neuron ports and physical neuron units, as well as enable corresponding physical neuron units that have been assigned to a certain virtual port index. This process essentially consists of a number of successive searching and adaptive resource allocation steps. Figure 17 shows the workflow of this adaptive physical neuron allocation process.

Once the V2P mapper receives the information about the number of virtual neurons (N_{vn}) and physical neurons (N_{pn}) from the ANN Controller, it will start to check the availability



Network Congtroller
Network Topology determined by Controller
Enter “Comprised” Mode
Number of Virtual Neurons: N_{vn}
Number of Physical Neurons: N_{pn}
No enough Physical Neurons for current topology
Error Detector
Virtual Neuron: i
i=1
Faulty Neuron String (FS)
0 0 1 0 0
1 0 0 0 0
Check the availability of Physical Neuron j, j=1
Used Neuron String ($USED$)
$FS[j]$=1?
No
j=j+1
i=i+1
Yes
Assign VN i to PN j:
$INDEX[j]$=i
$USED[j]$=1
i=N_{vn} ?
j=N_{pn} ?
i=N_{vn} && j<N_{pn}
V2P Mapping Done

Figure 17: Adaptive Physical Neuron Allocation Workflow

of all physical neurons (PNs). Beginning from the first physical neuron PN_1, if there is no fault information tagged in the Faulty Neuron String (FS) (i.e., $FS[1] \neq 1$) and no occupation information tagged in Used Neuron String (USED) (i.e., $USED[1] = 0$), it is shown that PN_1 is available and thus PN_1 will be assigned to the first virtual neuron (VN) port (i.e., $INDEX[1] = `0b00001'$). Correspondingly, the first bit in the Used Neuron String should be tagged ($USED[1] = 0 \rightarrow USED[1] = 1$) to reflect the occupation status of PN_1. Otherwise, if PN_1 is not available, the V2P mapper will go to the next physical neuron until it finds next available neuron PN_j which will be assigned to the current VN port i (i.e., $INDEX[j] = i$). The faulty physical neurons and unused ones will be deactivated by assigning a zero value (i.e., $INDEX[j] = 0$). This process will continue and be iterated until either all physical neurons have been "sold out" (i.e., $j = N_{pn}$) or all virtual neuron ports have been successfully designated (i.e., $i = N_{vn}$). The former case will make the system go into the "Compromised" mode, where the ANN system can still maintain certain level of operations without the need to stop and replace the whole system. The latter situation will create the desired ANN structure as ANN Controller specifies and disable all faulty neuron units and redundant ones.

Since the V2P Mapper is essentially a sequential searching and allocation process, the time overhead that it may introduce is completely in direct proportion to the searching space (i.e., the number of physical neuron units) and the frequency of conducting a V2P remapping. Either the stage transitions in ANN training/functioning processes or a newly detected faulty neuron will trigger a V2P remapping operation. Since our ARANN architecture integrates the hidden layer and output layer together by reusing homogeneous physical neuron units to meet the stringent area requirements of future portable applications, there are at least four major stage transitions that need to remap the V2P connections, as shown in Figure 18. Assuming the neural network structure is made up of a hidden layer with N_{hdn} neurons and a output layer with N_{out} neurons, as well as the training process involves Num_{input} input patterns and lasts for Num_{epoch} epochs, thus the overall introduced extra time overhead is:

$$T_{overhead} = [(N_{hdn} + \epsilon) \times 2 + (N_{out} + \epsilon) \times 2] \times (Num_{input} \times Num_{epoch} + f) \quad \text{cycles} \tag{4.4}$$

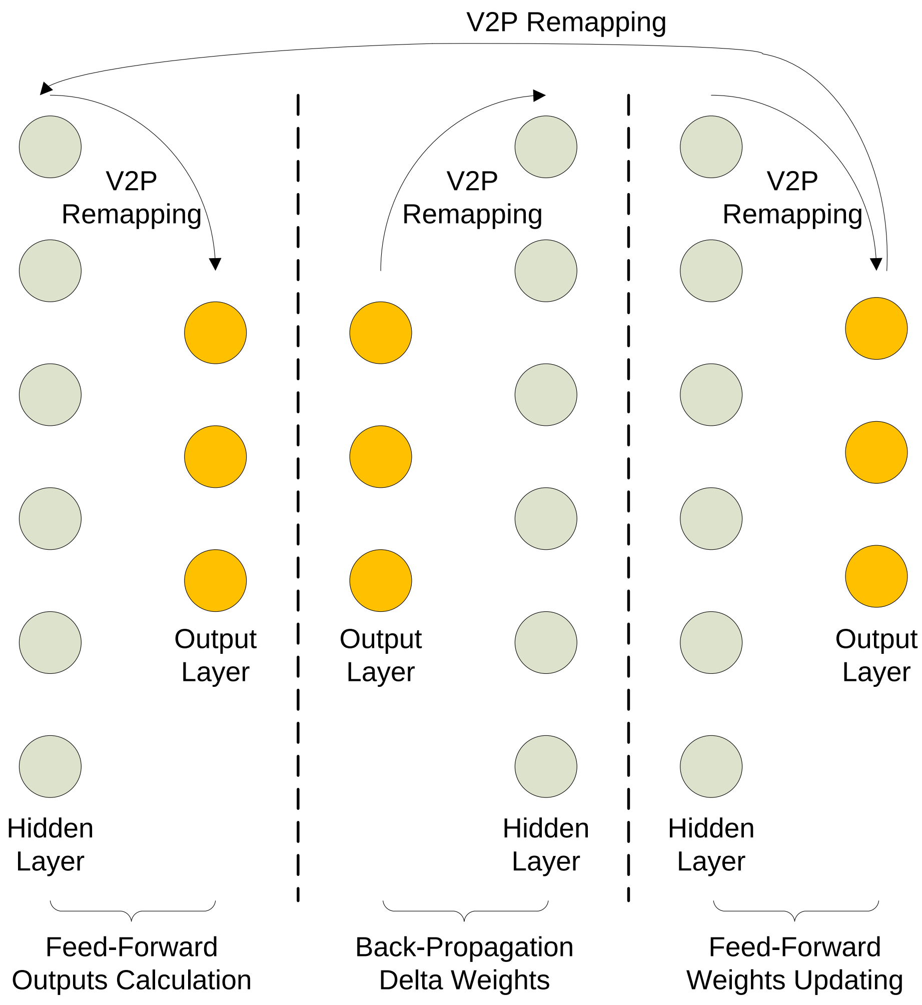


Figure 18: Stage Transitions and V2P Remappings during Neural Network Training Process

where f is the times when a new faulty neuron is detected, and the ϵ is a few extra cycles needed in a V2P remapping process besides the main searching and allocation steps. It is shown that the dedicated V2P mapping block will provide the most flexibility to ARANN and introduce affordable time cost only when applying a relatively small ANN structure (small numbers of hidden neurons and output neurons) onto a simple problem (i.e., small set of input patterns and small amount of epochs). Otherwise, the proposed V2P mapping block will cause non-negligible time overhead.

4.5.2 Cache-Accelerated Adaptive Physical Neuron Allocation

According to the discussion above, a dedicated V2P mapping block was proposed to determine the appropriate connections between virtual neuron ports and physical neuron units. When there are any changes associated with virtual neuron ports (i.e., stage transitions, topology adaptations) or any changes on the availability of physical neurons (i.e., a newly detected faulty neuron), such V2P mapper will be triggered to execute and adapt to a new V2P mapping scheme. This strategy makes the proposed ARANN architecture more flexible and adaptable to achieve large-scale, high-level fault tolerance and system optimization. However, the frequent involvement of V2P neuron remapping processes will introduce extra time cost, which could even counteract the benefits of flexible system adaptations when considering large neural networks for complex applications. Therefore, we would like to explore a better efficient solution with less time overhead than the current scheme.

Inspired by the hierarchical memory system in state-of-the-art computer architectures, we propose a Cache-Accelerated Adaptive Physical Neuron Allocation approach. The main idea is to utilize the temporal locality of V2P mapping solutions, that is, if at one point in time a particular V2P neuron mapping scheme is generated by the V2P mapping block, then it is likely that the same mapping scheme will be referenced again in the near future. This temporal locality exactly comes from the inherent characteristics of the ANN training/functioning process. As shown in Figure 18 and Equation 4.4, most of time cost comes from the repeatedly switching of V2P mapping schemes. That is, when ANN system switches from hidden layer to output layer, a new V2P mapping scheme is created to adapt to the current virtual neuron demands (for output layer). Similarly, when ANN system switches from output layer to hidden layer, another new V2P mapping scheme is established. However, these two remapping operations only need to be executed once, if the ANN topology has been fixed and there is no new faulty neuron unit detected during the whole training/functioning process. Therefore, we propose to add a cache-like register file into aforementioned V2P mapping block to store the most recently used V2P mapping scheme.

Figure 19 illustrates the architectural diagram of the proposed physically-tagged cache, which is used to store the most recently used V2P mapping schemes. In this case, we only

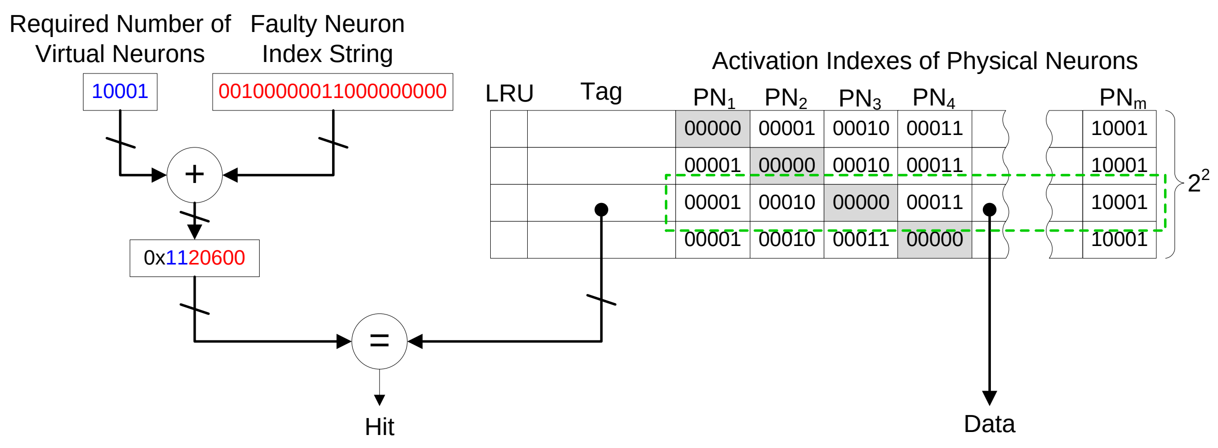


Figure 19: Architectural Diagram of Physically-Tagged Cache for Physical Neuron Allocation

show a simple example maintaining 4 mapping solutions. The *Data* part consists of the virtual neuron indexes associated with all physical neurons ($PN_1 \rightarrow PN_m$), where the gray filled cells represent the damaged neurons based on the information provided by the Error Detector. The *Tag* area keeps the unique addressing condition (i.e., the number of desired virtual neurons and the availability of each physical neuron specified by the Faulty Neuron Index String) for the corresponding V2P mapping scheme stored in the data part. The *LRU* field keeps track of which cache line (mapping scheme) was used when, to make sure the least recently used item will be discard and replaced if there is another new mapping scheme newly generated by the V2P mapper. In such implementation, every time a cache line is used, the "LRU age bits" of all other cache lines changes. With such a cache structure, if there is any change on the desired number of virtual neurons or the availability of physical neurons, the ANN system will firstly check the V2P mapping cache and then load corresponding mapping scheme immediately if the current addressing condition (i.e., concatenated blue and red bit strings in Figure 19) exactly matches one of tags. The V2P mapper will be triggered only if a completely new mapping solution is needed, i.e., no same one was used recently and can not be found in the V2P mapping cache. Given the distinct temporal locality of V2P mapping schemes in most ANN training/functioning processes, we can imagine that the time overhead can be reduced significantly using th proposed Cache-Accelerated V2P Mapping Block. The overall time cost can be represented in the following equation:

$$T_{overhead} = (N_{hdn} + N_{out} + 2\epsilon) \times (1 + f) + \xi \times (Num_{input} \times Num_{epoch}) \quad \text{cycles} \tag{4.5}$$

where ξ is quite a few cycles needed for cache access, which is usually only around 2 or 3 cycles. Comparing with Equation 4.4, it is shown that a significant amount of V2P remapping efforts have been optimized to simple cache access operations and considerable time overhead caused by V2P remapping have been eliminated accordingly. This cache integrated strategy is particularly useful and beneficial for a relatively stable ANN system (i.e., fixed topology without any need to change or optimize the number of desired neurons) used for a complex problem (i.e., a large set of input patterns and many epochs). On the other side, the cost of this strategy comes from its hardware implementation, a few extra logics used for the addressing tag comparison and data accesses of the cache as well as a small register file used for storing V2P mapping schemes.

4.5.3 Virtual-to-Physical Neuron Mapping Memory

In last two sections, we have presented two strategies to establish appropriate mapping mechanism between the virtual neuron ports specified by the ANN Controller and the homogeneous physical neuron units implemented on hardware, according to the currently desired number of neurons and the availability of physical neuron pool. The key component in both of aforementioned strategies is an adaptive V2P mapping block, which can be triggered to react to any change on either virtual neuron side or physical neuron side. Given the fact that a combination of the desired number of neurons and the present availability of physical neurons will generate a *unique* Virtual-to-Physical Neuron Mapping scheme based on the proposed V2P mapper. Thus, an intuitive way to facilitate the V2P mapping of ANN system is to compile a look-up table and maintain this table in a Read-Only Memory (ROM) or other storage devices.

The architectural diagram of the proposed V2P Mapping Memory is illustrated in Figure 20. The address of the memory is made up of a concatenated binary string of the

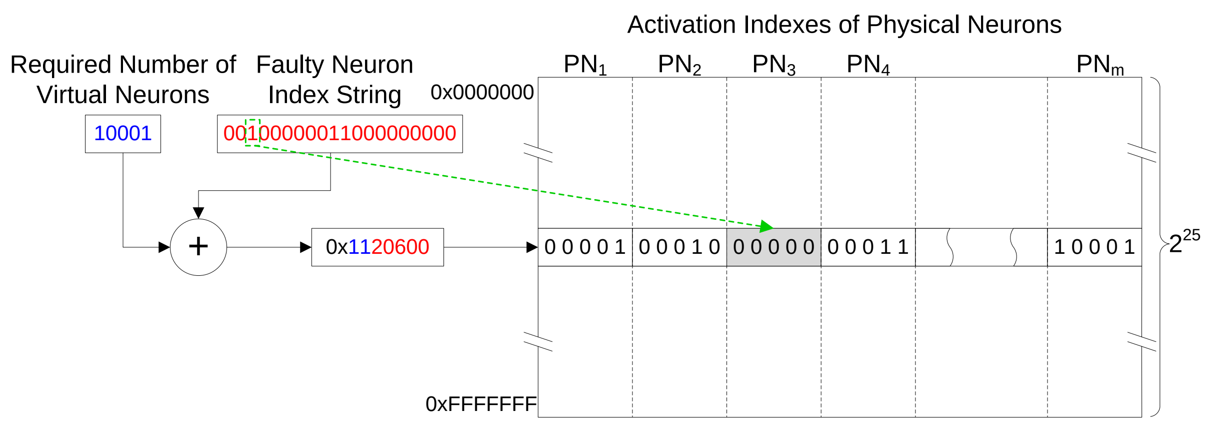


Figure 20: Architectural Diagram of Virtual-to-Physical Neuron Mapping Memory

desired number of virtual neurons and the Faulty Neuron Index String, which reflecting the availability of each physical neuron unit. The content of each memory line contains virtual neuron indexes (non-zero values) and deactivation signals (zero value) affiliated with all physical neuron units. Once such a V2P mapping table is compiled, it will provide ARANN the complete solutions no matter what or when any scale of V2P remapping is required. The system only needs to perform a number of simple memory access operations to quickly achieve virtual-to-physical neuron remapping. It is supposed to introduce the minimum time overhead.

Unfortunately, This V2P Mapping Memory strategy also has some remarkable drawbacks. Firstly, it demands an off-line computation to compile the V2P mapping lookup table, although this won't be too challenging given the computational capabilities of mainstream computers. Another major concern is the potential size of such a V2P Mapping Memory capable of accommodating the whole lookup table. Now, we would like to estimate the potential memory size to completely store all V2P mapping solutions.

Similar as aforementioned two strategies, we still assume the ARANN architecture owns a neuron pool which contains NUM_{neu} physical neuron units. Accordingly, the maximum number of virtual neurons used by ANN Controller is also NUM_{neu}, which demands at least $\lceil \log_2 NUM_{neu} \rceil$ binary bits to express all virtual neuron indexes. The Faulty Neuron Index

String also needs NUM_{neu} binary bits to represent the availability of each physical neuron. Therefore, the total memory size will be

$$SIZE = \underbrace{2^{\lceil \log_2 NUM_{neu} \rceil + NUM_{neu}}}_{\text{memory depth}} \times \underbrace{\lceil \log_2 NUM_{neu} \rceil \times NUM_{neu}}_{\text{memory width}} \tag{4.6}$$

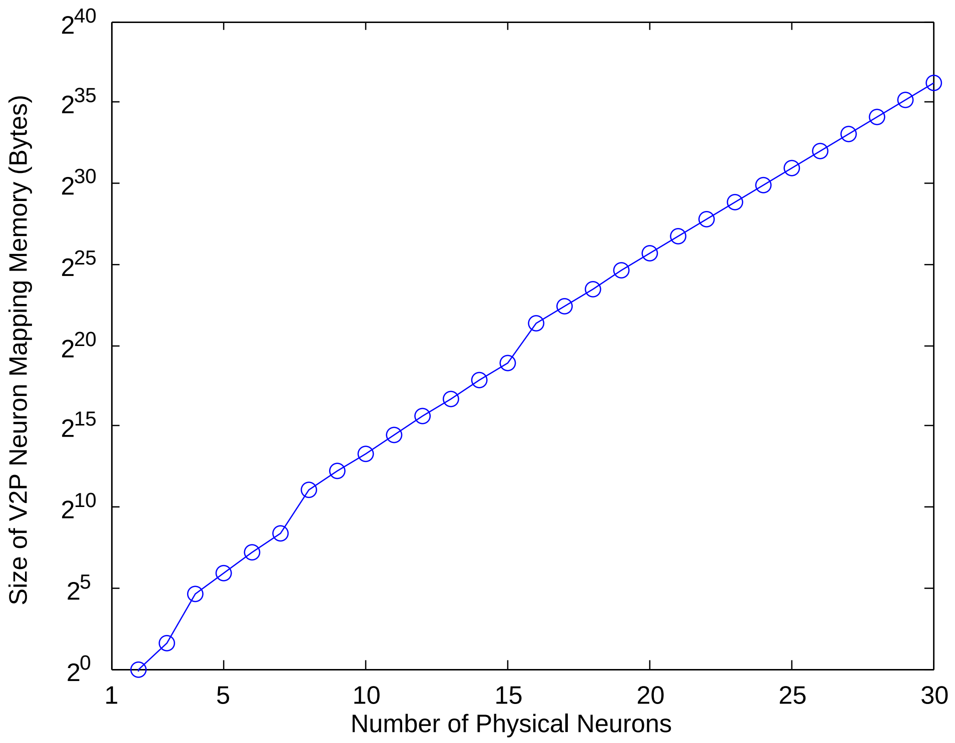
According to the equation above, for instance, if the size of physical neuron pool is 30, the total size of the V2P Mapping Memory will be

$$SIZE_{30} = 2^{\lceil \log_2 30 \rceil + 30} \times \lceil \log_2 30 \rceil \times 30 = 2^{35} \times 150 = 600 \times 2^{33}\text{bits} = 600\text{GB} \tag{4.7}$$

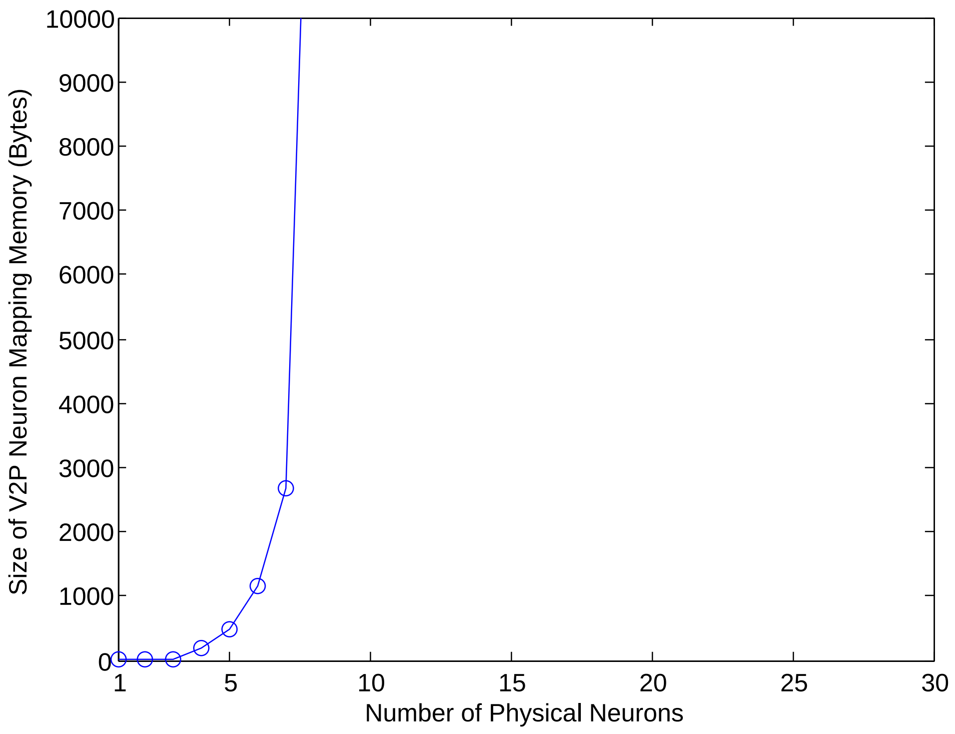
It is manifest that this memory size is not affordable for an ANN system with less than 30 neurons, not to mention for a complex ANN structure with even more neuron units. Equation 4.6 presents an exponentially increasing demand on the size of the V2P mapping memory, as the increase of involved neuron units. This trend can be illustrated in Figure 21.

4.5.4 Mask-Based Virtual-to-Physical Neuron Mapping Memory

Assuming one or more faulty neurons have been identified and reported within a neural network, in our previous study, the structural adaption can be very intuitive to isolate the faulty nodes and disconnect all their associated connections to other neurons (Figure 22) by remapping the connections between virtual neuron ports and physical neuron units. We can compile a V2P mapping lookup table to cover all possible mapping schemes no matter how many (virtual) neurons are needed by ANN Controller and no matter how many physical neuron units are damaged. The ARANN system can easily reconfigure itself to react to any change on structural topology or physical availability. However, it has been shown that the size of such a V2P mapping memory will increase exponentially as the expansion of neural network scale and will be unlikely affordable even for a moderate ANN system, as shown in Equation 4.6. For instance, the required memory size is 80KB for 10 neurons, 400MB for 20 neurons, 600GB for 30 neurons, and etceteras. The results also imply that such a direct V2P mapping lookup table capable of handling individual physical neuron units is definitely not an implementation-efficient approach due to a tremendous amount of storage



(a) (On the logarithmic scale)



(b) (On the normal scale)

Figure 21: Sizes of V2P Neuron Mapping Memory for Different Number of Physical Neurons

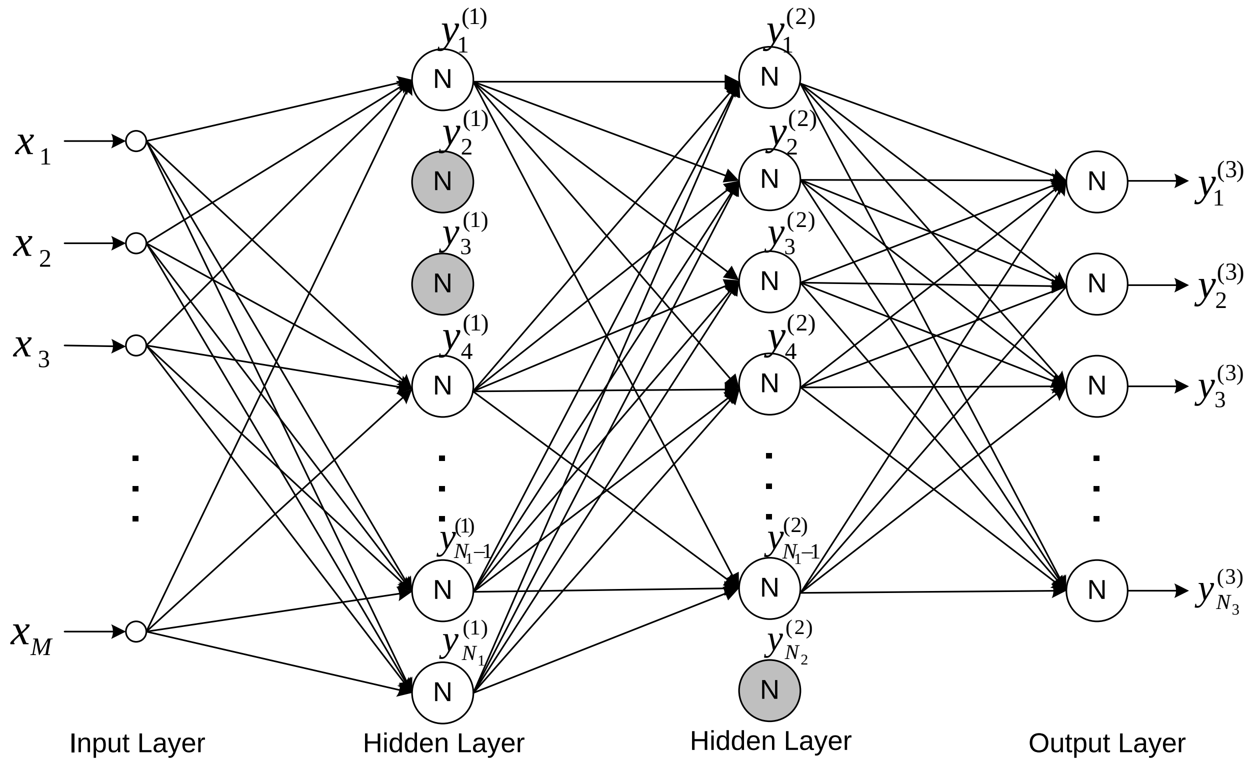


Figure 22: Reconfigured ANN Interconnection Topology By Isolating the Faulty Neurons

space needed to completely cover all possible remapping conditions and to fully utilize all remaining "healthy" neurons with least loss of performance.

Since the artificial neural networks are inherently self-adaptive learning systems, which are achieved by adjusting the network weights according to the provided training data set (for *supervised learning*), disconnecting/disabling only a few faulty neural nodes has little effect on the overall learning performance of the neural network. In order to reduce the size of V2P neuron mapping memory to an acceptable level, we propose a mask-based V2P mapping strategy. The key idea is to reduce the granularity from a single neuron to a group of neurons set by the mask and then to investigate the V2P mapping solutions on a mask basis rather than the individual neuron basis. In this way, the design complexity and hardware overhead can be significantly reduced by trading-off the fine-grained granularity.

Instead of preparing a separate V2P neuron mapping scheme that uses all remaining available resources for each faulty case considering the scale and location of faulty neurons, we propose to group several neurons together as a *faulty mask*. No matter which neuron(s) within such a group cannot work properly, the whole group will be disabled and disconnected from the network. Correspondingly, only one V2P mapping scheme is needed for all faults occurring on the neurons within this group. Figure 23 shows two examples with different sizes of faulty mask: the mask size of the former one is 3 neurons and the latter one is 2 neurons. Comparing these two examples, for the same faulty neurons, it is shown that with a larger faulty mask, a smaller number of various V2P mapping schemes are required, while more "healthy" neurons would be discarded causing the loss of resource and performance.

In order to explore an optimal design solution with considerable system performance and affordable hardware requirements, we conduct a rigorous quantitative study on the effects of faulty mask size on the ANN system performance. Assuming the neural network contains N homogeneous physical neuron units and the probability of the occurrence of a faulty neuron is p, the overall neural resource utilization will be:

$$\begin{aligned} T_{total} =& \mathbf{C}(N,1) \times p^1 \times (1-p)^{N-1} \times (N-1) + \mathbf{C}(N,2) \times p^2 \times (1-p)^{N-2} \times (N-2) \\ & \ldots + \mathbf{C}(N,N-1) \times p^{N-1} \times (1-p)^1 \times 1 + \mathbf{C}(N,N) \times p^N \times (1-p)^0 \times 0 \\ =& \sum_{i=1}^{N} \mathbf{C}(N,i) \times p^i \times (1-p)^{N-i} \times (N-i) \end{aligned} \tag{4.8}$$

Similarly, given all N neurons have been partitioned into M mask blocks, the neural resource utilization of mask-based structural adaption would be:

$$\begin{aligned}T_{mask} =& \Big\{ \mathbf{C}(M,1) \times \Big[1-(1-p)^{\frac{N}{M}}\Big]^{1} \times \Big[(1-p)^{\frac{N}{M}}\Big]^{M-1} \times (M-1) + \\ & \mathbf{C}(M,2) \times \Big[1-(1-p)^{\frac{N}{M}}\Big]^{2} \times \Big[(1-p)^{\frac{N}{M}}\Big]^{M-2} \times (M-2) + \\ & \ldots + \mathbf{C}(M,M) \times \Big[1-(1-p)^{\frac{N}{M}}\Big]^{M} \times \Big[(1-p)^{\frac{N}{M}}\Big]^{0} \times 0 \Big\} \times \frac{N}{M} \\ =& \frac{N}{M} \sum_{i=1}^{M} \mathbf{C}(M,i) \times \Big[1-(1-p)^{\frac{N}{M}}\Big]^{M-i} \times \Big[(1-p)^{\frac{N}{M}}\Big]^{i} \times (M-i)\end{aligned} \tag{4.9}$$

where

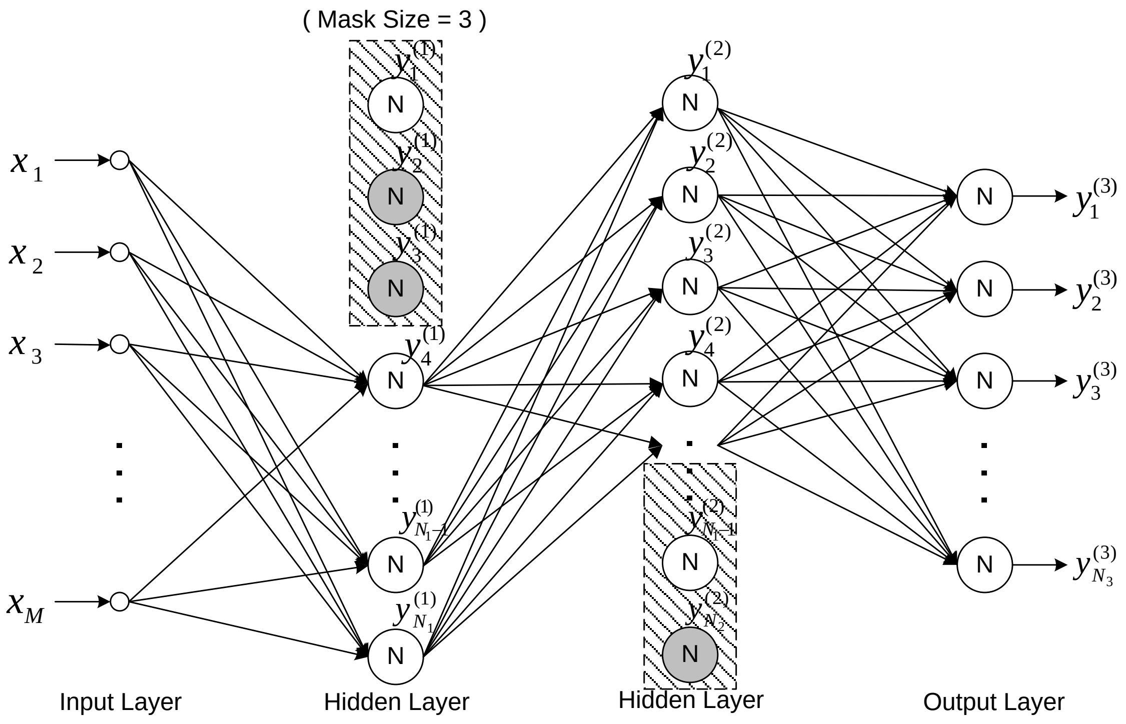
$$P_{mask} = 1-(1-p)^{\frac{N}{M}} \tag{4.10}$$

represents the probability of a faulty mask block with at least one faulty neuron (i.e., maybe all neurons within this block are all damaged due to unexpected errors). Correspondingly, $1 - P_{mask}$ gives the probability of a completely "healthy" block where all neurons work properly and are not physically damaged.

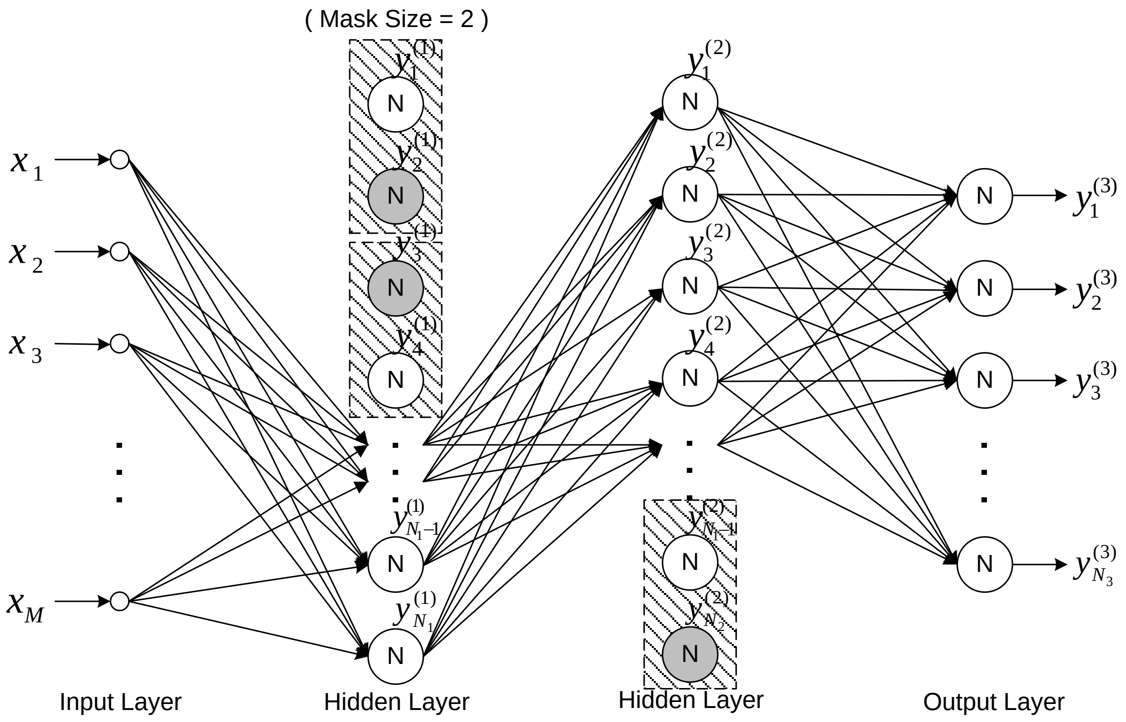
Given the ideal situation where the remaining healthy neurons can be fully used by the new reconfigured neural network topology, we would like to investigate the effective utilization of neuron resources by mask-based structural adaption approach, which can be expressed in following way:

$$Ratio = \frac{T_{mask}}{T_{total}} \tag{4.11}$$

As shown in Figure 24, it is not surprising that the normalized effective utilization ratios of mask-based methods increase gradually as more mask blocks have been partitioned within the whole network. When there are N mask blocks, each of which actually only consists of one single neuron, such extreme case will be identical to the ideal case where all healthy neurons will be fully used and all faulty neuron combinations will be considered. It is also demonstrated that the faulty mask-based method is much more effective when the probability of the occurrence of a damaged neuron unit is small. That means, the abandonment of healthy neurons in the faulty mask only causes limited impacts on the system performance and the proposed mask-based solution can be much more effective when the hard faults do not occur frequently, vice versa.



(a) (Mask Size = 3 Neurons)



(b) (Mask Size = 2 Neurons)

Figure 23: Mask-Based Reconfigured ANN Structural Topology

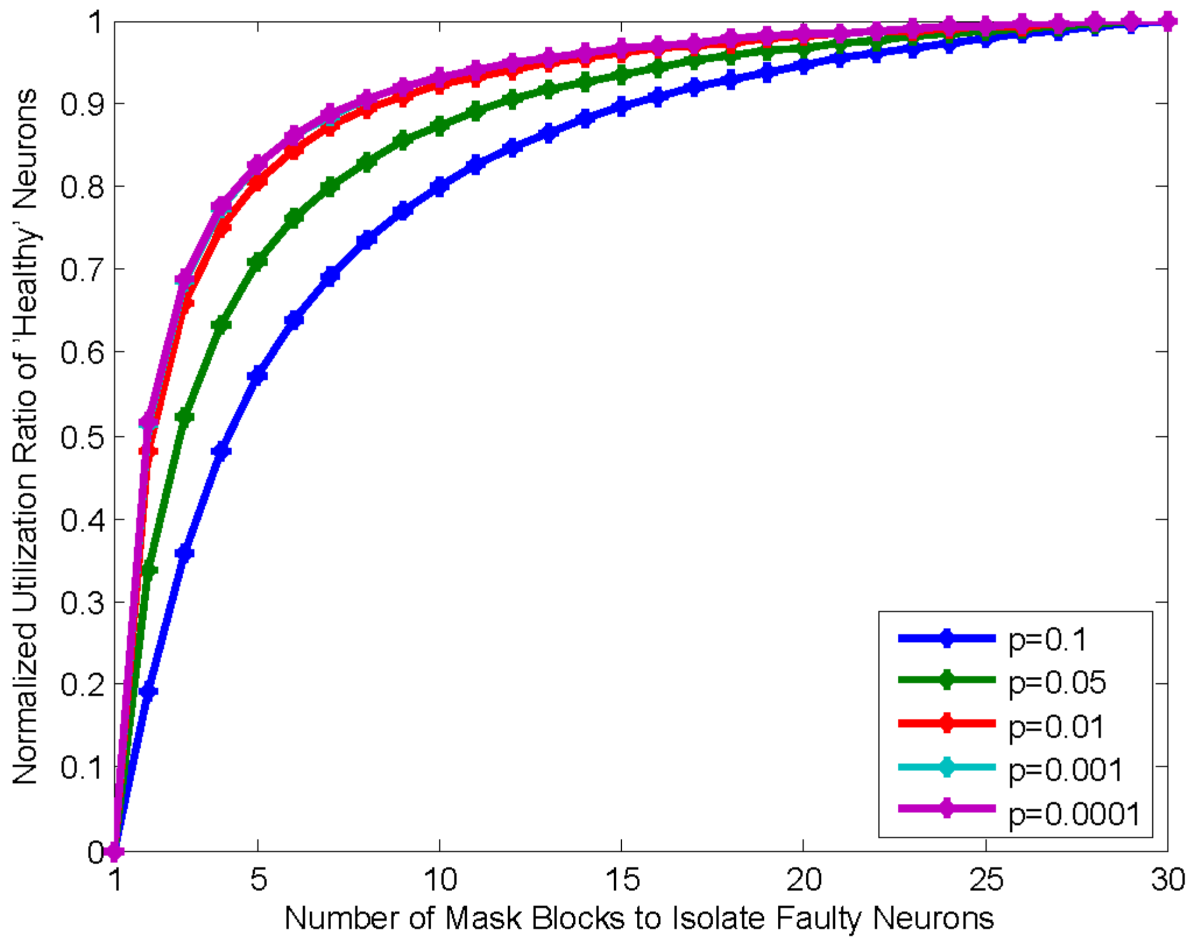


Figure 24: Normalized Utilization Ratio of 'Healthy' Neurons with Different Probability (p) of Hard Faults

So far, we have discussed the efficacy and efficiency of the proposed mask-based neural topology adaptation capable of dealing with any amount of faulty neurons. It has been demonstrated that such method can achieve considerable reduction of design complexity with maintaining the proper functionality of the whole neural network by isolating the faulty neurons and possibly their adjunct neighbors from all other healthy neurons. However, the aforementioned discussion is based on the assumption that the system only reacts to the potential risks by simply removing all faulty nodes and constructing a new platform based on all remaining available resources. Given the decoupled virtual neurons and physical neurons implemented in the ARANN, this assumption is not always the case because the actual required number of neurons (virtual neurons) can be less than the number of available physical neurons. If the physical neuron pool can still provide some available neuron units even after disabling those faulty neurons, the new V2P mapping scheme can be viewed as an alternative of previous mapping scheme by swapping the original connections to faulty neurons and the connections to previously unused healthy neurons. In this case, the mask-based structural adaptation may probably not hurt the system performance at all, while providing reduced design complexity. In what follows, we will investigate and discuss the efficacy and efficiency of an augmented version of mask-based structural adaptation which takes into account the available "spare" neurons in the physical neuron pool.

For instance, as shown in Figure 25, three spare neurons are activated during the topology reconfiguration and swapped into the neural network to partially compensate the loss of those faulty neurons. Comparing Figure 23(a) and Figure 25, the only difference is whether new neurons are introduced to the network or not. The availability of a few spare physical neuron units can help to maintain the overall performance of neural networks with damaged neuron nodes, but also consumes extra hardware resource and power to deploy redundant units on board. Therefore, determining the appropriate number of physical neurons implemented on hardware involves a synergic and systematic process to find a balanced trade-off between the performance benefits of redundant logics and their expensive cost in resources. In Figure 26, we illustrate a thorough analysis on the performance benefits of different levels of redundancy. Since our V2P mapping strategy will automatically search and involve available physical neuron units to the most extent according to the number of virtual neurons specified by

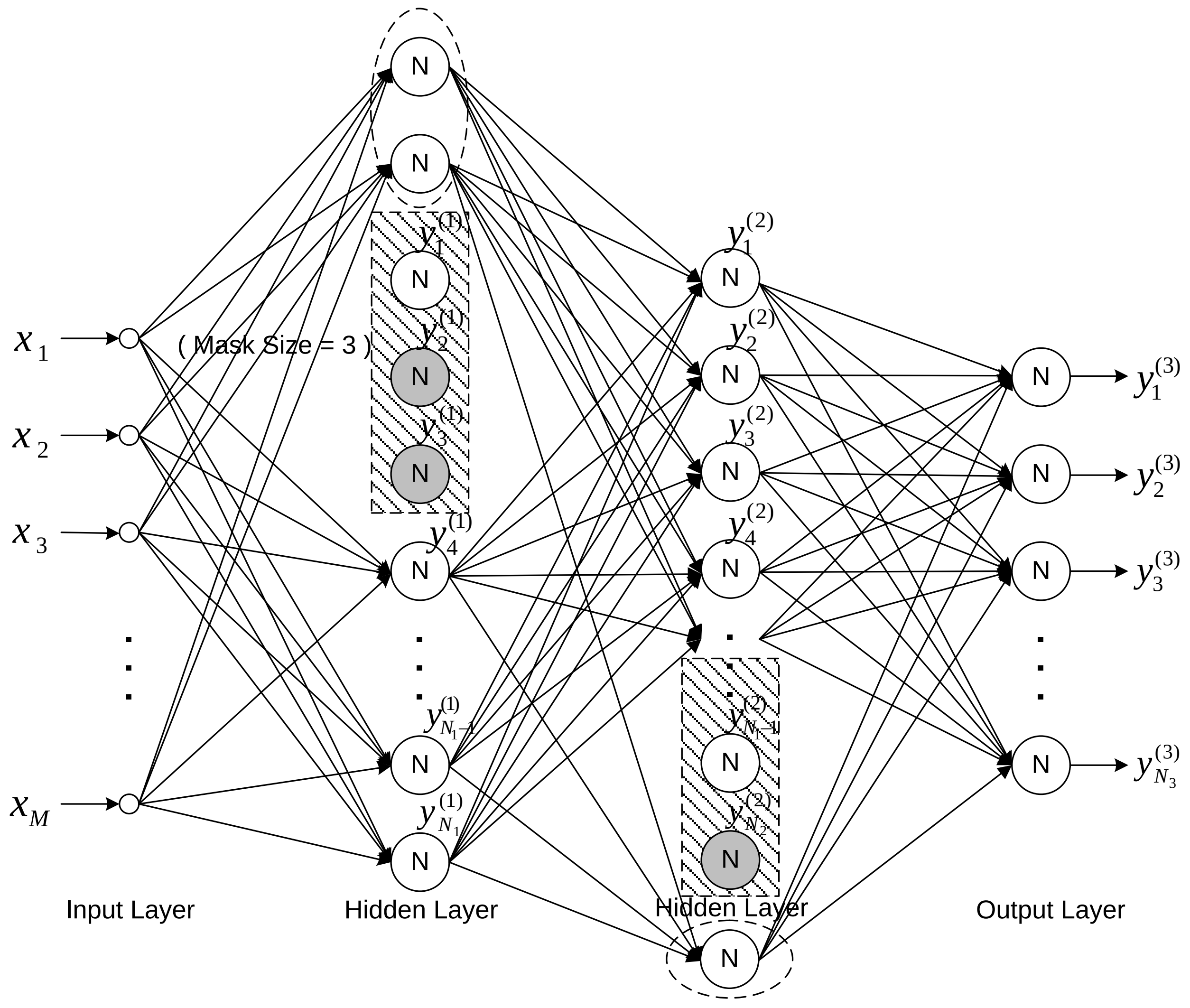


Figure 25: Reconfigured ANN Interconnection Topology with Backup Neurons

the ANN Controller, assuming ARANN's physical neuron pool is well designed and most of applications need to use majority of physical neuron units provided, thus we can expect there are likely at most 5 spare neurons available to be swapped into the neural network when one or more neuron units are damaged. Given incremental numbers of spare physical neurons available on board, the normalized resource utilization ratio can be further increased, comparing with the fault-tolerant efficiency shown in Figure 24. It is also shown that the incremented benefits of deploying more physical neurons than the number of (virtual) neurons truly needed in ANN for most applications are gradually becoming marginal when 5 spare neurons have been involved into the system reconfiguration.

Based on the discussion and analysis above, it has been demonstrated that the coarse-grained mask-based structural adaptation is able to maintain an appropriate level of learning capability and functioning performance of the ANN systems, with significantly reduced design complexity. On a basis of grouped neuron units, we now can further optimize our previously proposed V2P neuron mapping memory strategy in section 4.5.3. Retrieving the idea of V2P mapping memory, it is essentially required that we compile a complete lookup table recording all possible V2P mapping schemes on a basis of individual physical neurons. The desired number of virtual neurons and the detected locations of damaged neurons will behave as a unique indexing address to access the corresponding memory line, which contains all index information representing the connections between virtual neuron ports and specific physical neuron units. Once we partition all physical neurons on a mask basis, the original *Faulty Neuron Index String* is transformed into a new *Faulty Mask Index String*, combing with the desired number of virtual neurons, which will be used as a new form of address to locate one specific V2P mapping scheme. Accordingly, all V2P neuron mapping schemes currently stored in the memory are also generated on a basis of mask-based neuron groups. Figure 27 illustrates the improved Mask-Based Virtual-to-Physical Neuron Mapping Memory. Assuming a mask size of 3 neurons. When a faulty neuron is detected, the availability bit of its affiliated mask in the Faulty Mask Index String will be marked to "1" and all neurons within this mask will be disabled, as pointed out by the green lines in the figure. Comparing Figure 27 with Figure 20, we can observe that the main change is the indexing scale of the memory and the corresponding mask-based V2P schemes stored in the memory.

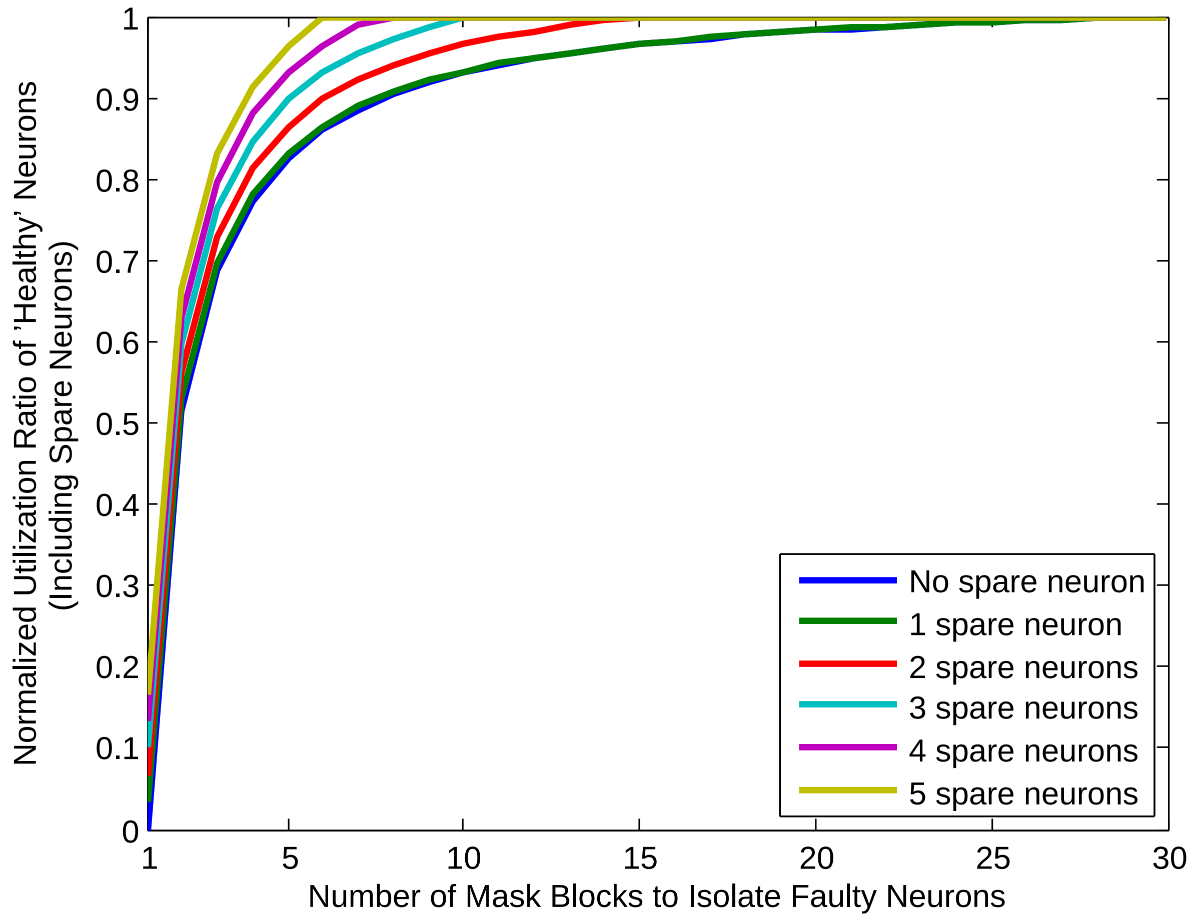


Figure 26: Normalized Utilization Ratio of 'Healthy' Neurons with Different Number of Backup Neurons

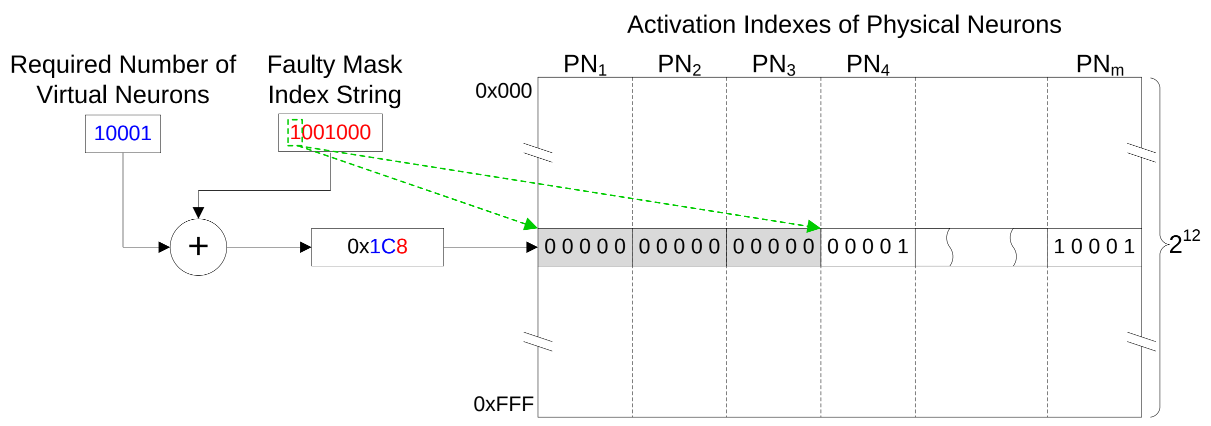


Figure 27: Architectural Diagram of Mask-Based Virtual-to-Physical Neuron Mapping Memory

For the example of a V2P mapping memory for 20 neurons, it needs at least 25 bits in the indexing address. In contrast, the new mask-based V2P mapping memory only needs 12 bits for memory address (including 7 bits used to represent the availability of seven masks out of 20 neurons). The required size of V2P mapping memory has been reduced from 400MB to 50KB. It is expected that we can implement a very affordable V2P neuron mapping memory using a slightly larger mask size, without too much loss of ANN system performance as we proved before.

Similar as Equation 4.6, with Num_{mask} masks introduced to perform structural adaptation on a basis of grouped physical neurons, the required size of V2P Neuron Mapping Memory will be determined in the following way:

$$SIZE = 2^{\lceil \log_2 NUM_{neu} \rceil + NUM_{mask}} \times \lceil \log_2 NUM_{neu} \rceil \times NUM_{neu} \tag{4.12}$$

where $NUM_{mask} = \lceil \frac{NUM_{neu}}{\text{Mask Size}} \rceil$. Considering various mask sizes, we further investigate the effects of the proposed mask-based strategy on the total size of V2P mapping memories. Without loss of generality, here we only illustrate five different mask sizes in Figure 28.

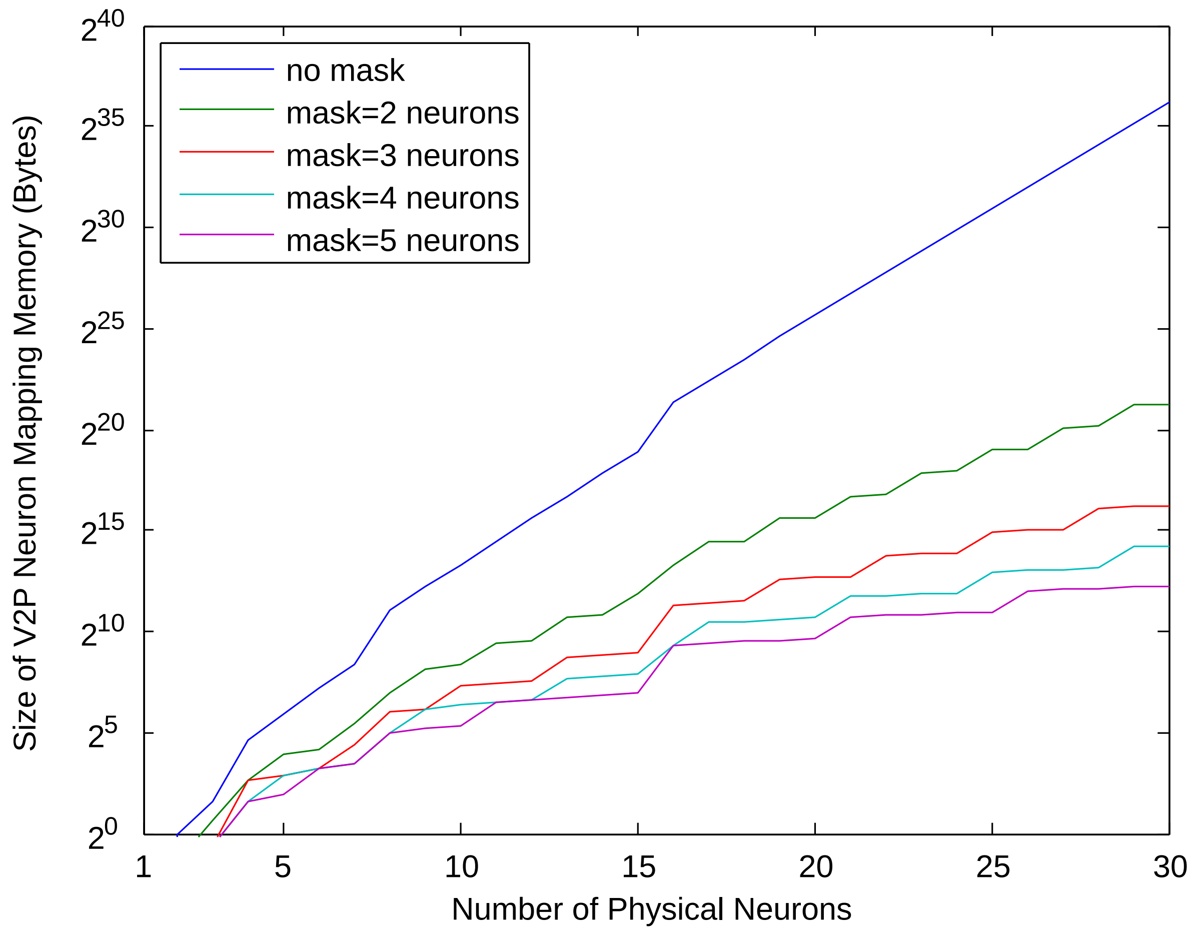


Figure 28: Sizes of V2P Neuron Mapping Memory for Different Physical Neuron Numbers and Mask Sizes

Compared with the tremendous amount of memory spaces needed to provide complete V2P mapping and ANN system reconfiguration solutions on a fine-grained basis of individual neuron manipulations, the mask-based approach can provide effective neuron resource utilization in a very efficient way. That is, such mask-based approach can achieve comparable performance (as shown in Figure 26) as the extremely resource-consuming fine-grained neuron mapping approach with significantly less hardware resource by several orders of magnitude (as shown in Figure 28). For example, if 30 physical neurons are clustered into six groups (mask size is 5 neurons), only a 37.5KB memory is needed to maintain a light-weight version of V2P neuron mapping lookup table, in contrary to the 600GB storage space required to store all fine-grained V2P neuron mapping solutions.

4.5.5 Comparisons and Considerations of V2P Mapping Schemes

Until now, we have presented four design strategies to implement the Virtual-to-Physical (V2P) neuron mapping, a critical component within the ARANN architecture, and to effectively facilitate ARANN's autonomous reconfiguration no matter when a new ANN topology is needed or a neuron unit is physically damaged while on duty. It is clear that these four design options have distinct characteristics in design complexity, resource requirement, time overhead, and applicability to various scales of problems. In this section, we would like to conduct a comprehensive analysis and comparison on the proposed four design strategies: 1) Adaptive Physical Neuron Allocation ("V2P Mapper"), 2) Cache-Accelerated Adaptive Physical Neuron Allocation ("V2P Mapper w/ Cache"), 3) Virtual-to-Physical Neuron Mapping Memory ("V2P Memory"), and 4) Mask-Based Virtual-to-Physical Neuron Mapping Memory ("Mask-based V2P Memory"). The specific features considered in the comparison and corresponding results are listed in Table 5.

According to Table 5, it is clearly shown that there isn't a perfect design choice and all these four design strategies have their own advantages and limitations. The *V2P Mapper* strategy is able to provide the most flexibility and generate optimal V2P remapping schemes using a dedicated logic block. However, it has the largest time overhead among all of four strategies and thus it is meaningful only when the ANN is applied to a simple, small-

Table 5: Comparisons of V2P Neuron Mapping Design Strategies[a]

Features	V2P Mapper	V2P Mapper w/ Cache
Slice LUTs[b]	969	1424
Memory	N/A	N/A
Time Overhead[c] **(cycles)**	$92(Num_{input}Num_{epoch} + f)$[d]	$92(1 + f) + 2Num_{input}Num_{epoch}$
Neuron Utilization (%)	100	100
Features	**V2P Memory**	**Mask-based V2P Memory**
Slice LUTs	N/A	N/A
Memory	600GB	37.5KB
Time Overhead (cycles)	$8(Num_{input}Num_{epoch} + f)$	$8(Num_{input}Num_{epoch} + f)$
Neuron Utilization (%)	100	~ 85

[a] All results are obtained based on an ANN with 30 physical neuron units (30 hidden neurons and 6 output neurons).
[b] Number of Slice LUTs is obtained from Xilinx ISE 11.1 based on Virtex-5 XC5VLX110T FPGA.
[c] The values within the formulas, which represent the total cycles needed to perform various operations (e.g., exhaustive searching, memory access, or register latching), are obtained based on current ARANN implementation. They may be slightly different depending on specific implementation strategies.
[d] $Num_{input}Num_{epoch}$ represents the repeated times of a single Feed-Forward Back-Propagation training period; f represents the times of changing ANN's topology caused by either newly detected faulty neuron or structural optimization purpose.

scale problem. As the enhanced version of the V2P Mapper, *Cache-Accelerated V2P Mapper* strategy can significantly alleviate the time overhead issue by keeping the most recently used V2P mapping schemes instead of recalculating them every time. However, the effectiveness of this cache structure highly relies on the cache hit rate, which is determined by the value of f, the probability of requests on changing ANN's topology caused by either newly detected damaged neurons or structural optimization purpose. That is, if the ANN system is extremely unreliable or it needs to be frequently adapted to fit different applications, the cache miss rate is relatively high and thus the time overhead will be still remarkable. Besides that, the enhanced V2P Mapper with cache consumes a little more hardware resource than the standard V2P Mapper. On the other side, given the fact that the combination of the desired neuron number and the current availability of each physical neuron can be projected to a unique V2P mapping scheme, we propose to pre-compile a V2P mapping lookup table and a specific V2P mapping scheme can be easily located and accessed if there is a request on ANN system adaptation. Although the *V2P Memory* strategy is capable of providing the same V2P mapping solution as the *V2P Mapper* with significantly less time overhead, it has to consume a huge amount of storage space that sometimes is even unfeasible. Therefore, this strategy is particularly useful for a small ANN structure applied to a very complicated problem (i.e., large Num_{input} or Num_{epoch}). Inspired by ANN's inherent fault-tolerant and non-sensitive (to parameter changes) characteristics, we proposed a mask-based network topology adaptation strategy. Instead of manipulating each neuron individually, we can reconfigure ANN's structure on a basis of coarse-grained grouped neurons. The quantitative analysis has shown that, if an appropriate mask size is selected, the mask-based topology adaptation can achieve over 80% neuron utilization ratio (refer to Figure 24 and 26). Applying the mask-based neuron clustering idea onto the aforementioned V2P Memory, we present a light-weight *Mask-based V2P Memory* structure, which can achieve comparable neuron utilization as the original V2P Memory strategy but only needs a very affordable memory space. According to the discussion above, the specific applicability of each V2P mapping design strategy has been summarized in Table 6. Since we only investigated and implemented a small-scale ANN system for a relatively simple biomedical application (see section 3.3) and we also assume a relatively low defective probability for our ANN system,

Table 6: Applicability of V2P Neuron Mapping Design Strategies

Requirements	**V2P Mapper**	**V2P Mapper w/ Cache**	**V2P Memory**	**Mask-based V2P Memory**
Scale of ANN (i.e., number of neurons)	large	large	small	moderate
Complexity of Problem (i.e., Num_{input}, Num_{epoch})	low	high	high	high
Frequency of ANN Reconfiguration (i.e., f)	low	low	high	high
Performance Requirement (i.e., neuron utilization rate)	high	high	high	low

thus the *Cache-Accelerated V2P Mapper* seems to be a good design choice in our experiments. Therefore, if not mentioned, we all use the V2P Mapper with Cache in following experiments.

5.0 ARANN IMPLEMENTATION CHALLENGES AND SOLUTIONS

5.1 ARITHMETIC REPRESENTATION

Since the available resource on FPGAs is always one of most challenging issues researchers are concerned about, and usually it is also the most direct bottleneck that implementing complicated functional modules on FPGAs, such as neural network structure, determining the most appropriate data precision and efficient arithmetic representation format becomes one of the important choices when implementing ANNs on FPGAs. It is agreed that a higher data precision means fewer quantization errors in the final implementations, while a lower precision leads to much simpler designs with higher speed, smaller area, and lower power consumption. Although 32-bit/64-bit floating-point formats (FLP) defined in IEEE-754 standard [123, 124] have been widely used in both general-purpose microprocessors, high-end embedded systems, and mostly all of software implementations, it has been demonstrated that using lower precision FLP or fixed-point (FXP) formats can significantly reduce hardware resource consumption (e.g., less area use on FPGAs), with a certain level of precision loss [108]. Thus, a format-dependent, precision-reduced efficient implementation of neural networks on FPGAs can result in completely different outputs from the same architecture implemented in software using IEEE FLP formats. This phenomenon is called the *area versus precision* design tradeoff [250], which including the selection of data format, the appropriate balance between the precision required to perform network properly and meet accuracy requirements, and the size and cost of FPGA resource consumption.

Holt and Baker [108] claimed that most neural network hardware designs implemented limited precision integer or binary computation, while most research on neural network algorithms and applications use single or double precision floating-point simulations. In order to provide an accurate vision about the influence of limited precision computation on the neural network algorithms, they investigated the minimum precision required for a class of benchmark classification problems (i.e., NetTalk, Parity, Protein, and Sonar) and found that the 16-bit fixed-point (1-bit sign, 3-bit integer bits and 12-bit fractional bits) was the minimum allowable precision without diminishing an ANN's capability to learn these benchmark problems.

Motivated by the noticeable design challenges faced by the implementation of floating-point operations on reconfigurable FPGA platforms, Ligon III *et al.* [165] deployed an IEEE single precision floating-point adder and multiplier on old generation Xilinx FPGA — 4020E, 4062XL, and 40250XV, and explored the research question if, and when, FPGAs may become practical for use in algorithms requiring floating-point computations in the context of drastically increasing densities. They also showed that the space/time requirements for 32-bit FXP adders and multipliers are still less than those of their 32-bit FLP counterparts, although those floating-point implementations had successfully fit in a Xilinx 4020E FPGA board and achieved a performance of 40 MFLOPS [165].

Recently, Draghici [62, 63, 64] proposed a more mathematically rigorous approach to verify the validity of the limited precision from a theoretical point of view. He [63] relates the "difficulty" of a given classification problem characterized by the minimum distance between patterns of different classes to the weight range/precision necessary to ensure the existence of at least one valid solution. Draghici [64] further proved that, neural networks with integer weights in the range of $[-p, p]$ is able to solve any classification problems for which the minimum Euclidian distance between two patterns from opposite classes is $1/p$. It was shown that the number of bits is limited by $m \times n \times log(2pD)$ where m is the number of patterns, n is the dimensionality of the space, p is the weight range and D is the radius of a sphere including all patterns. Draghici's studies provide an important theoretical guideline to achieve a balanced trade-off between the hardware implementation cost and accuracy of neural networks by selecting appropriate weight precisions.

5.1.1 Floating-Point Format

Floating-Point (FLP) numbers have an advantage of being able to cover a much larger dynamic range compared to fixed-point numbers. However, correspondingly, it also brings much more complexity for the implementation in hardware.

The IEEE-754 standard [123, 124] specifies a representation for single and double precision floating-point numbers. It is currently the standard that is used for real numbers on most computing platforms. Floating-point numbers consist of three parts: sign bit, mantissa, and exponent. In the IEEE-754 format, the mantissa is stored as a fraction (f), which is combined with an implied one to form a mantissa ($1.f$) such that the mantissa is multiplied by the base number (two) to an exponent e, as shown in Equation 5.1 and 5.2, single and double precision, respectively [14]:

$$X = (-1)^s \cdot 1.f \cdot 2^{e-127} \tag{5.1}$$

$$X = (-1)^s \cdot 1.f \cdot 2^{e-1023} \tag{5.2}$$

The IEEE standard specifies a sign bit, an 8-bit exponent, and a 23-bit mantissa for a single precision floating-point number, as shown in Figure 29(a). Double precision floating-point has a sign bit, an 11-bit exponent and 52-bit mantissa, as shown in Figure 29(b). Since the mantissa is normalized to the range $[1, 2)$ there will be always be a leading one on the mantissa. By implying the leading one instead of explicitly specifying it, a single bit of storage could be saved, but it does raise the complexity of floating-point implementations.

5.1.2 Fiexd-Point Format

A Fixed-Point (FXP) number represents a real data type for a number that has a fixed number of digits after the radix point (i.e., typically a decimal point "."). FXP numbers are particularly useful for representing fractional values, usually in base 2 or base 10, when considerable computation performance is required with limited hardware resources or floating-point unit (FPU) is not available. Actually, in a majority of the commercially available

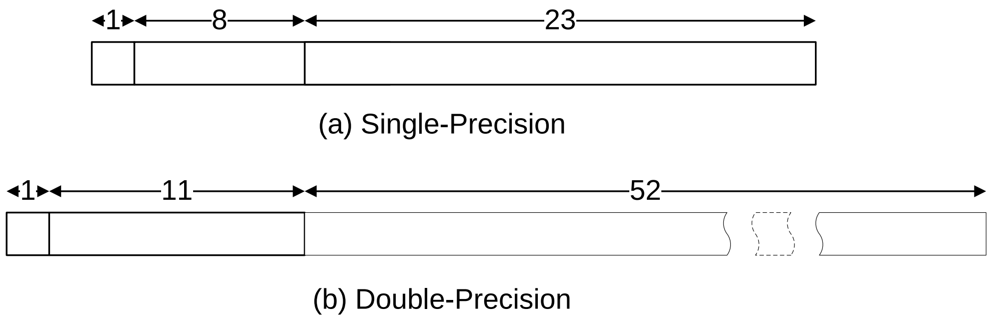


Figure 29: IEEE Floating-Point Numbers

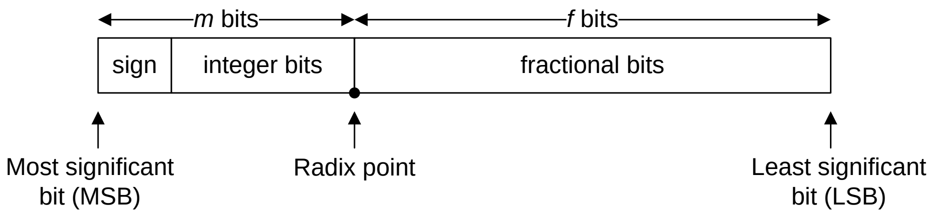


Figure 30: Format of A Fixed-Point Number

processors on the market today, there is no hardware support for floating-point arithmetic due to the cost the extra silicon imposes on a processor's total cost [203]. Especially for most low-cost embedded microprocessors and microcontrollers, which have taken up about 55% of all CPUs sold in the world (according to Semico Research Corporation, Phoenix, AZ), the fixed-point representation and arithmetic show a significantly competitive advantage.

A fixed-point number is essentially an integer that is scaled by a certain factor. Binary fixed-point numbers are most frequently used, because their rescaling operations can be easily implemented as fast bit shifts. To represent a fractional number in binary fixed-point format, it needs to be viewed as two distinct parts — the integer content and the fractional content, and is defined with the following notation:

$$Q\ m\ .\ f \tag{5.3}$$

where the Q prefix declares a fixed-point format, m represents the number of magnitude or integer bits, and f describes the number of fractional bits. The number of integer bits (m) plus the number of fractional bits (f) yields the total number of bits used to represent the number (as shown in Figure 30). The sum of $m + f$, known as the Word Length (WL), usually corresponds to a specific processor or a given design architecture (typically 8-bit, 16-bit, or 32-bit). For example, $Q6.10$ describes a number with 6 integer bits and 10 fractional bits stored as a 16-bit two's complement binary [279]. Since the entire word is a two's complement binary, a sign bit has been implied within integer bits (m). Without loss of generality, a binary fixed-point type in two's complement format, with f fractional bits and a total of b bits, has a lower bound of $-(2^b - 1)/2^f$ and an upper bound of $(2^{b-1} - 1)/2^f$, where 2^f is the scaling factor and $b - 1$ is the number of bits not counting the sign bit.

Another important issue designers are always concerned about for the fixed-point format is the representation resolution. The resolution of a fixed-point variable ϵ, is determined by the number of fractional bits (f) according to the following equation:

$$\epsilon = \frac{1}{2^f} \tag{5.4}$$

Contrarily, the least number of fractional bits f required for a particular computational resolution can be determined in the following way:

$$f = log_2\left(\frac{1}{\epsilon}\right) \tag{5.5}$$

Given the number of fractional bits must be an integer value, Equation 5.5 can be further revised using the *ceiling* function that round the result to the next largest integer:

$$f = ceiling\left(log_2\left(\frac{1}{\epsilon}\right)\right) \tag{5.6}$$

Moreover, it is specially worth pointing out a recently proposed "dual FXP" representation [73], which augments the flexibility of classic fixed-point formats with an additional "exponent" bit representing the position of the radix point. As shown in Figure 31, with two preselected position settings of the radix point, this format provides two possible ranges and precisions the number can actually represent, given the value at the "exponent" bit. However, in this study, we still focus on the classic fixed-point representation to make our

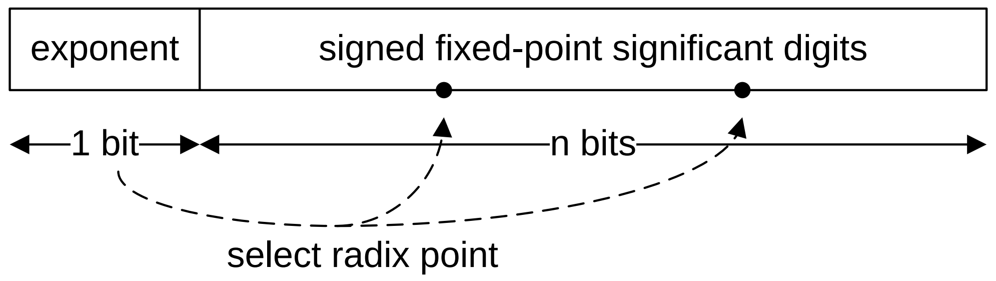


Figure 31: Format of A Dual FXP Number [250]

reconfigurable neural network platform easily adapt to any existing computer architecture and applications. It is expected that the training performance and operation accuracy can be improved with the flexibility of radix point, range and precision provided by such dual FXP format in the future work.

5.1.3 Comparisons Between FLP and FXP Formats

Specifically targeting multilayer perceptrons trained using the error backpropagration algorithm (MLP-BP) neural networks, Antony W. Savich and his colleagues [250] implemented the MLP-BP network based on several FXP and FLP arithmetic formats. The effects of data representation and numeric precision on overall resource consumption, network convergence, and training performance were exhaustively compared and discussed. According to their study, the FXP-based implementation is always smaller in area, compared to a FLP-based implementation with similar precision and range by approximately a factor of two. Moreover, the FXP representation is somewhat faster in clock rate and significantly better in latency than its FLP counterpart. Besides that, the FXP format also provides better training convergence results over FLP formats. Considering the future transplantable requirements and spatial regularity desired by scale-independent implementations, we use the fixed-point $Q6.10$ (i.e., 1-5-10) format in our following studies.

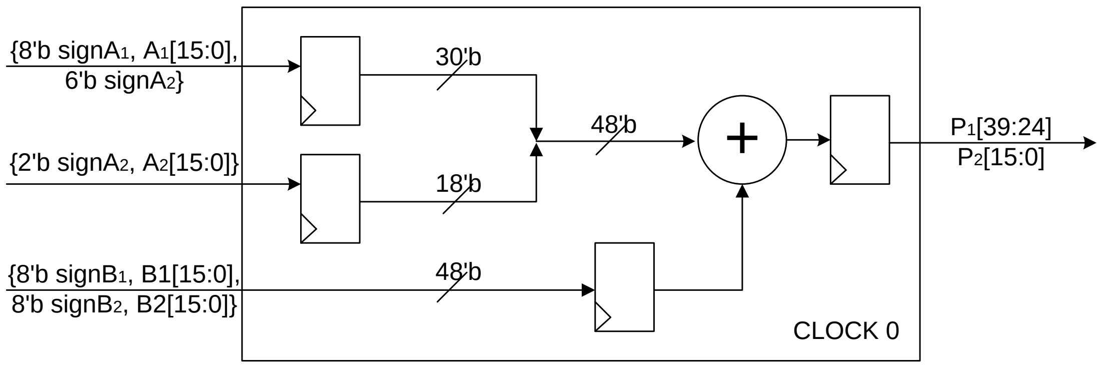


Figure 32: Two 16-bit SIMD Dynamic Adder/Subtractor

5.2 COMPACT MULTI-PURPOSE NEURONS

For Xilinx Virtex-4 or Virtex-5 series FPGAs, the XtremeDSP Digital Signal Processing DSP48/DSP48E slices have been integrated as new elements, which were referred to as Application Specific Modular Blocks (ASMBL) architecture. The purpose of this model is to deliver off-the-shelf programmable devices with the best mix of logic, memory, I/O, processors, and digital signal processing [300, 302]. The DSP48E slice supports many independent functions, including multiply, multiply add, three-input add, barrel shift, bit-wise logic functions. They also could be cascaded to form wide math functions. To achieve the highest resource saving, we will reuse as many slices as possible. One of the best choices is to fully utilize the versatile capability of DSP48E. The frequently used full-length shifters, addition and subtraction, multiplication, as well as multiply-add/sub operations all will be built using DSP48E slices.

5.2.1 Two Input 16-Bit SIMD Dynamic Adders/Subtracters

The DSP48E slice can be easily configured as a full 48-bit dynamic adder/subtractor, where the 30-bit input A and 18-bit input B are concatenated to form a 48-bit operand and the other operand directly comes from 48-bit wide input C. In our fixed-point add/subtract operations, either a 16-bit adder ($A_i + B_i$) or a 16-bit subtractor (($A_i - B_i$)) needs to be used within a neuron. Such smaller operand addition/subtraction in the DSP48E slice requires sign extension all the way up to the 48th bit for the C input C[47], and A input A[29]. However, there is noticeable waste in operand resources because of its merely 1/3 valid occupation of all available operand bits. Fortunately, the new DSP48E slice introduces the Single Instruction Multiple Data (SIMD) mode and is capable of being split into four 12-bit adders/subtractors or two 24-bit adders/subtractors with carry out signal per segment. The SIMD mode can be used efficiently to support two independent 16-bit dynamic adder/subtractor simultaneously with appropriate sign extension, as shown in Figure 32. By changing the ALUMODE parameter, the specific addition and subtraction operations can be dynamically selected based on the effective operations decided by neuron functional decoder.

Adding two's complement numbers requires no special processing if the operands have opposite signs: the sign of the result is determined automatically because the result can not go beyond any of the two operands. Otherwise, if 2 two's complement numbers with the same sign (both positive or both negative) are added, the overflow may occur due to our fixed-point numeric representation restricted to 16 bits of precision: any carry to the (nonexistent) 17th most significant bit (MSB) will be ignored. The rules for detecting overflow used in this study is "*overflow occurs if and only if the results has the opposite sign*" [67]. That is,

- If the sum of two positive numbers yields a negative result, the sum is overflowed.
- If the sum of two negative numbers yields a positive result, the sum if overflowed.
- Otherwise, the sum has not overflowed.

In other words, the result is not overflowed if the carry INTO the MSB equals the carry OUT OF the MSB. Once an overflow is detected, the result will be rounded in "Round Toward Zero" mode, in which all numbers beyond the representation range will be rounded to either the lower bound of $(100000.0000000000)_2$ (*i.e.*, -2^5) or the upper bound of $(011111.1111111111)_2$ (*i.e.*, $(2^{15}-1)/2^{10}$).

5.2.2 16-Bit Two's Complement Multiplication

The two's complement multiplier inside the DSP48E slice support two 25-bit $\times$ 18-bit, two's complement inputs and produces a 43-bit, two's complement result. Cascading of multipliers

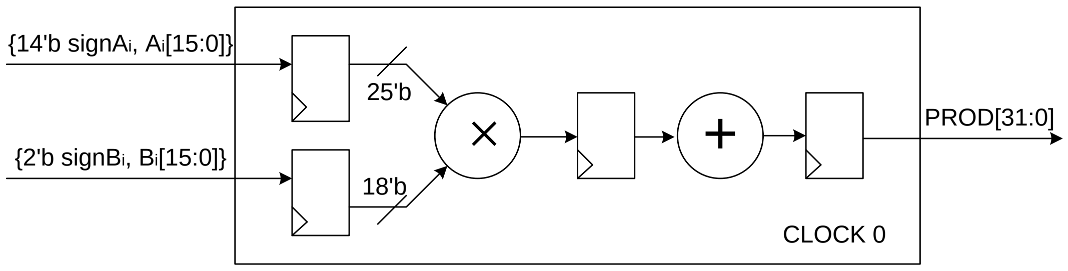


Figure 33: 16-bit Two's Complement Multiplier

to achieve larger products is implemented by a internal 17-bit right-shifted cascaded bus input to the adder/subtractor to right adjust partial products by appropriate bits [302]. In this study, however, we only need to deal with signed 16×16 multiply operations — $A_i \times B_i$, the operands of which can be easily mapped onto the input ports of DSP48E and thus generate corresponding signed, two's complement, 32-bit results.

In our design, since we use the 16-bit fixed-point representation format (as shown in Section 5.1), it is necessary to truncate the multiplication outputs in order to comply with the following arithmetic operations. Similar as Add/Sub operations, we also use the "Round Toward Zero" mode for the multiplication, with a little more complicated overflow detection mechanism, as shown in Table 7.

5.2.3 Squared Errors

Error calculation is an important aspect of any neural network, no matter whether the neural network is supervised or unsupervised. Researchers have investigated many error calculations in an effort to find a calculation with a short training time appropriate for the network's application. The most popular error function is the sum-of-squared error, which is calculated by looking at the squared difference between the target value and what the current network predicts for each training pattern. Formally, such error can be expressed in the following equation:

$$E = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{C} \left(t_{ij} - \hat{t_{ij}} \right)^2 \tag{5.7}$$

Table 7: Rounding Rules for 32-bit Multiplication Results

$$\underbrace{s}_{\text{Sign}}\ \ \underbrace{a_{10}\ a_9\ a_8\ a_7\ a_6\ a_5}_{\text{Overflow Bits}}\ \ \underbrace{a_4\ a_3\ a_2\ a_1\ a_0}_{\text{Valid Bits}}\ \ .\ \ \underbrace{b_0\ b_1\ \cdots\ b_8\ b_9}_{\text{Valid Bits}}\ \ \underbrace{b_{10}\ b_{11}\ \cdots\ b_{18}\ b_{19}}_{\text{Rounded Bits}}$$

Sign	Integer		Fraction		Rounding
Bit	Overflow Bits	Valid Bits	Valid Bits	Rounded Bits	Results
0	$\exists a_i = 1$	xxxxx	xxxxxxxxxx	xxxxxxxxxx	011111.1111111111
0	000000	$a_4a_3a_2a_1a_0$	$b_0b_1\cdots b_8b_9$	xxxxxxxxxx	$0a_4a_3a_2a_1a_0.b_0b_1\cdots b_8b_9$
1	111111	$a_4a_3a_2a_1a_0$	$b_0b_1\cdots b_8b_9$	xxxxxxxxxx	$1a_4a_3a_2a_1a_0.b_0b_1\cdots b_8b_9$
1	$\exists a_i = 0$	xxxxx	xxxxxxxxxx	xxxxxxxxxx	100000.0000000000

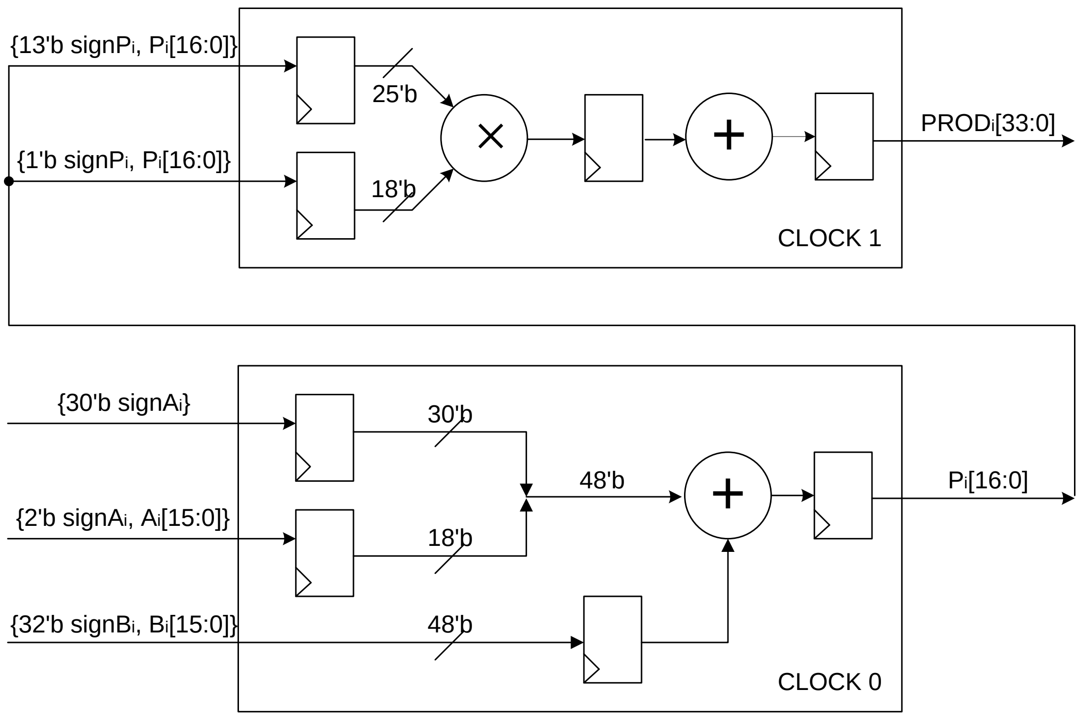


Figure 34: Sum of Squared Errors Based on DSP48E

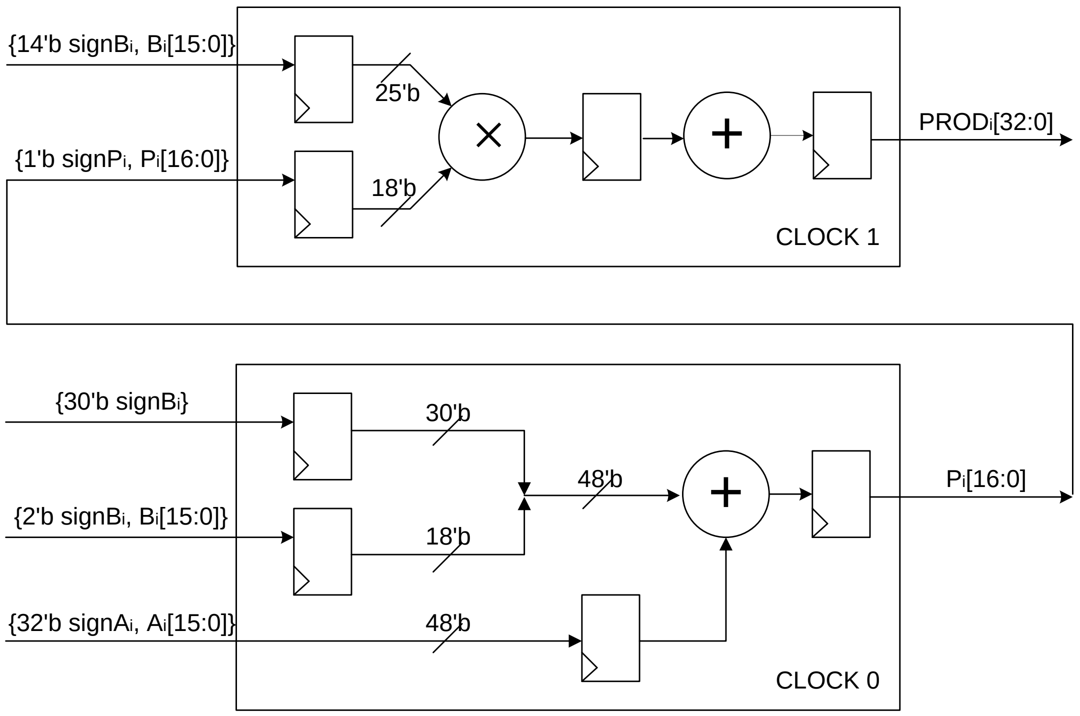


Figure 35: DSP48E-Based Subtract-and-Multiply Operation: (A-B) x B

where N is the total number of training cases, C is the number of outputs, t_{ij} is the target value for the ith training case and the jth network output, and $\hat{t_{ij}}$ is the corresponding predicted value by the current network for that case.

Thus, it is shown that the operation of squared error difference $(A_i - B_i)^2$ is intensively used in the training process of a neural network. With the functional flexibility of DSP48E slices, such a squared error operation can be conveniently processed on one DSP48E slice in two adjacent cycles. In the first cycle, like the conventional 16-bit subtraction operation described above, two operands A_i and B_i are sent to calculate their difference. As shown in Figure 34, the generated difference is directly sent back to both of two input ports as the multiplier and multiplicand. It is worth pointing out that the difference between target value and predicted value is maintained in 17-bit precision as the intermediate result, so that there is no need to enforce a rounding process on it and it also improves the calculation accuracy with the least precision loss.

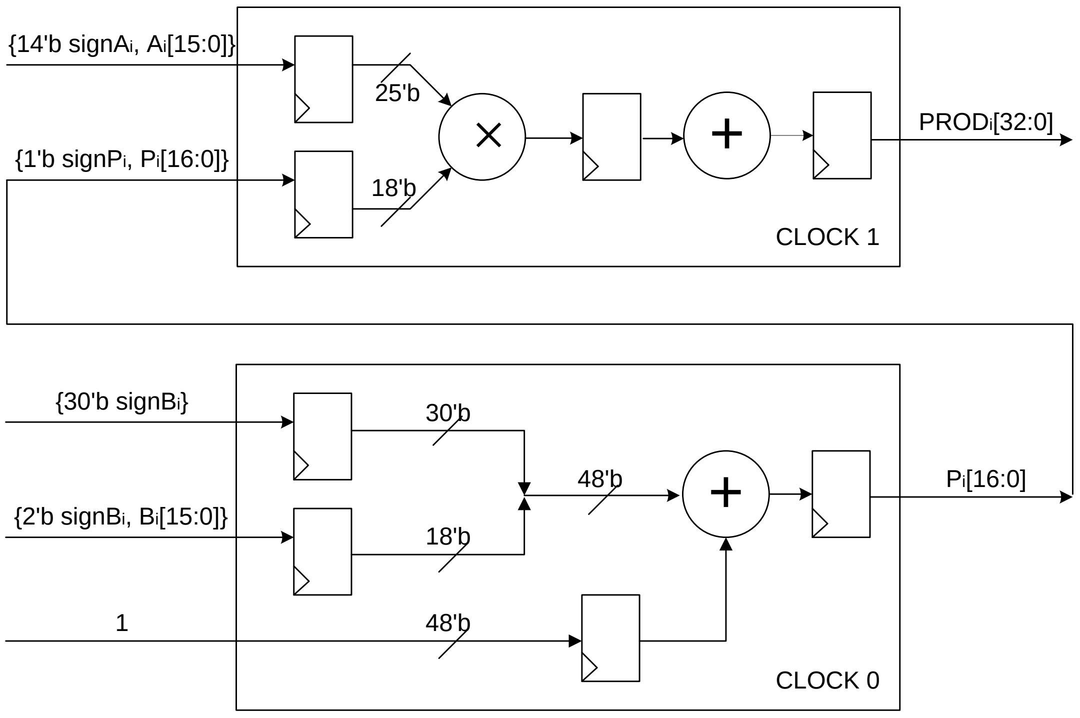


Figure 36: DSP48E-Based Subtract-and-Multiply Operation: A x (1-B)

5.2.4 Subtract-and-Multiply Operations

Two other important calculations involved in the back-propagation process of neural network training have very similar behaviors: both of them perform a subtraction operation first and then execute a multiplication on two operands respectively — $(A_i - B_i) \times B_i$ and $A_i \times (1 - B_i)$. Thus, it is still possible and area-efficient to implement them on a single DSP48E slice using two adjacent cycles. The specific design scheme is shown in Figure 35 and Figure 36.

5.2.5 Multiply-Accumulate (MAC) Operations

One of the most complicated issues affiliated with artificial neural network is its inherent intricate interconnections among a large number of computational nodes — neurons, however, which is also the key reason that makes ANN capable of emulating the powerful recognitive and analytical capability of the human brains. Within the feed forward training or pattern recognition process, the neurons have to collect all the information transferred from all of neurons in the former layer, that is usually accomplished by the multiply accumulate unit (MAC) according to a sum-of-products function: $\sum(A_i \times B_i)$.

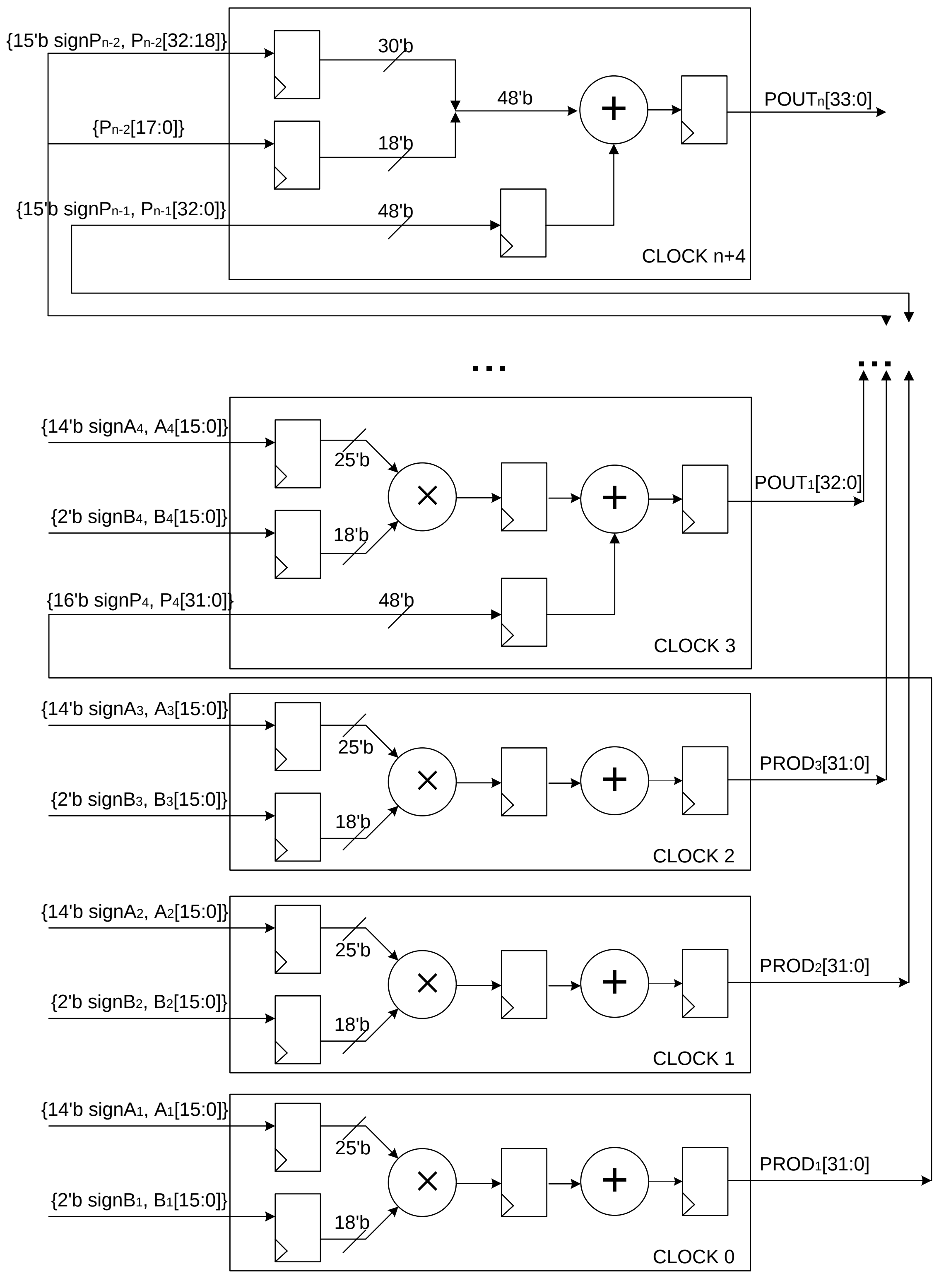
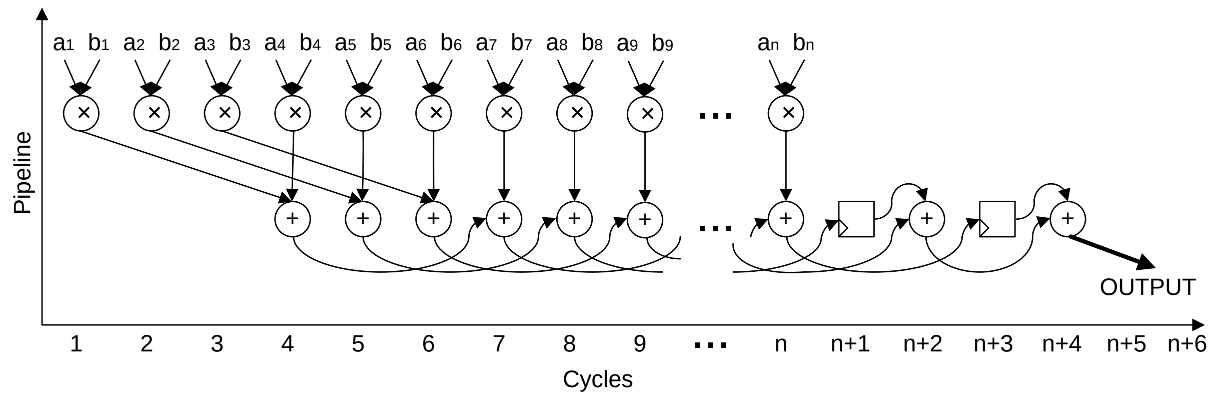


Figure 37: DSP48E-Based Multiply-Accumulation



a1 b1 a2 b2 a3 b3 a4 b4 a5 b5 a6 b6 a7 b7 a8 b8 a9 b9 an bn
Pipeline
OUTPUT
1 2 3 4 5 6 7 8 9 n n+1 n+2 n+3 n+4 n+5 n+6
Cycles

Figure 38: Operation Data Flow and Pipeline Scheduling of Multiply-Accumulation

A multiply-add block is supposed to receive the plurality of inter-neuron transferred information and corresponding predetermined weights, and to be able to provide an output representing a sum of each received value multiplied by the constant of a corresponding weight. The integrated 25×18 multiplier followed with a 48-bit adder in the DSP48E slice can facilitate such operations without introducing any extra latency. In order to achieve the highest performance and largest throughput, the computational nodes are dedicatedly designed to feed one pair of operands every cycle and send intermediate results back to the input ports of accumulation adder in a pipelining way. As shown in Figure 38, given a series of operands are fed into the DSP48E slices continuously one per each cycle, except the first three pairs of operands, the multiply-add functional blocks are fully utilized in every cycle and the throughput could be approximately the ideal case of one output per cycle. Considering execution cycles needed by the adder (2 cycles) and the multiply-adder (3 cycles), the total execution cycles of a sum-of-products operation are $n + 6$ for n pairs of operands. The specific circuitry intra-connection and corresponding data path timing are demonstrated in Figure 37.

5.3 IMPLEMENTATION STRATEGIES OF ACTIVATION FUNCTIONS

In bio-inspired artificial neural networks, one main computational stage within a neuron is to handle all information transferred from former layer using Activation Functions (AFs), that is usually an mathematical abstraction representing the rate of action potential firing in the cell. Its simplest form is a binary switch, that is, the neuron is either firing or completely not. Similarly, a straight line with positive slope can represent the increase in firing rate that occurs as input current increases. Usually, a normalizable sigmoid activation function is widely used in multilayer perceptrons in the form of a hyperbolic tangent: the model stays at a stable state – zero until it starts to receive the input current, when the firing rate increases quickly at first, but gradually approaches an asymptote at the 100% firing rate. It restricts the applied input to lie within the specified range of (0,1) and then determine the corresponding outputs. The two common forms of this function are

$$\text{Continuous Log-Sigmoid Function:} \quad \varphi(n) = \frac{1}{1+e^{-\beta n}} \tag{5.8}$$

where β is a slope parameter, and

$$\text{Continuous Tan-Sigmoid Function:} \quad \varphi(n) = \frac{2}{1+e^{-2n}} - 1 \tag{5.9}$$

We choose the former Log-Sig activation function in the following study, because of its relatively simple derivative calculation, which is helpful for deducting the weight updates in many training algorithms. The derivative is given by:

$$\frac{d\varphi(t)}{dt} = \varphi(t)[1-\varphi(t)] \tag{5.10}$$

It is well agreed that the implementation of sigmoid activation functions and their corresponding derivatives in software is relatively direct and even simpler given some built-in mathematical libraries. However, it is not the case when implementing them in hardware, particularly targeting FPGAs, since many design issues need to be considered carefully [241]:

- The arithmetic modules, such as x^y and e^x, are not synthesis-friendly and can not be synthesized with desired performance.

- The hardware implementation of the divider is extremely resource-hungry, long-latency, and may cause significant performance bottleneck.

Since the direct implementation for non-linear sigmoid activation functions is very expensive, many researchers have worked on practical approaches to approximate sigmoid functions with simple FPGA designs. Among all of such efforts, two methods are widely used — Lookup Table (LUT)-based approach and Piecewise Linear (PWL) Approximation. Both of them are elaborated and compared below and another improved LUT-based design is proposed to better accommodate the hardware properties of FPGAs.

5.3.1 LUT-Based Approach

The Lookup Tables (LUTs), particularly the LUTs with 4–6 bits of input, are the key components in modern FPGAs. It is much more computationally efficient to replace a runtime calculation with a simple array indexing operation [312]. The savings in terms of processing time can be significant, since usually accessing a value from memory is much faster than undergoing an "expensive" computation, especially for some certain extremely complicated equation expansion. However, sometimes, the LUTs can be noticeably resource-hungry components gobbling up logic cells and memories on FPGAs, if the computation requires a moderately high degree of precision.

The sigmoid activation functions are exactly good cases that can be implemented using LUTs by means of discrete value, to overcome the design difficulties discussed above. Since the 16-bit fixed-point data representation format is used in this design and a precision of 16 bits needs to represents the inputs and results of LUT, then $2^{16} \times 16 = 1$Mbits LUT is need. It will consume a large amount of on-chip logic area and access time, which may affect the speed of computation.

5.3.2 BRAM-Based Approach

Given the considerable logic resource consumption of LUT-based on-chip realization of sigmoid activation functions, we propose to use a ROM-like scheme for the LUT purpose, leveraging the plenty of built-in Block RAMs available on FPGAs. Furthermore, in order to further optimize the resource usage efficiency, we only consider the higher 10 bits of input values and simply remove 6 less significant bits of original inputs, as shown below:

$$\underbrace{s}_{\text{Sign}} \quad \underbrace{a_4\ a_3\ a_2\ a_1\ a_0}_{\text{Integer Bits}} \quad . \quad \underbrace{b_0\ b_1\ b_2\ b_3}_{\text{Fraction Bits}} \quad \underbrace{b_4\ b_5\ b_6\ b_7\ b_8\ b_9}_{\text{Discarded Fraction Bits}}$$

And then the trimmed input values are formatted as the addressing indices and the expected 16-bit results of sigmoid functions are stored in associated lines.

With such modification, the computational resolution is reduced from 0.001 $(0.0000000001)_2$ to 0.0625 $(0.0001)_2$ within limited loss of accuracy. However, the memory usage is significantly reduced from $2^{16} \times 16 = 1$Mbits to $2^{10} \times 16 = 16$Kbits, which now is able to fit into a single 18-Kbit BRAM block out of 120 such blocks in total.

5.3.3 Piecewise Linear Approximation Approach

Among all the efforts to approximate the sigmoid functions with simple calculations, the Piecewise Linear Approximation (PWL) method stands out due to its very simple arithmetic operations and relatively efficient hardware implementation. The sigmoid function is approximated by five adjacent linear segments called "pieces" [241]. The specific expression of these five segments is shown in Equation 5.11 [108]:

$$f(x) = \begin{cases} 0, & \text{if} \quad x \leq -8 & \text{(region 1)} \\ \frac{8-|x|}{64}, & \text{if} \quad -8 < x \leq -1.6 & \text{(region 2)} \\ \frac{x}{4} + 0.5, & \text{if} \quad |x| < 1.6 & \text{(region 3)} \\ 1 - \frac{8-|x|}{64}, & \text{if} \quad 1.6 \leq x < 8 & \text{(region 4)} \\ 1, & \text{if} \quad x \geq 8 & \text{(region 5)} \end{cases} \tag{5.11}$$

The number of segments required in the PWL approximation of the activation function can be further adjusted according to the complexity of the problem to be solved [312]. Note that if the coefficients for each linear segment representation are chosen to be powers of two, the hardware implementation efficiency can be further augmented using a series of regular shift and add operations [312]. Such PWL approximation method has been well used in many implementations of neuron activation functions [298].

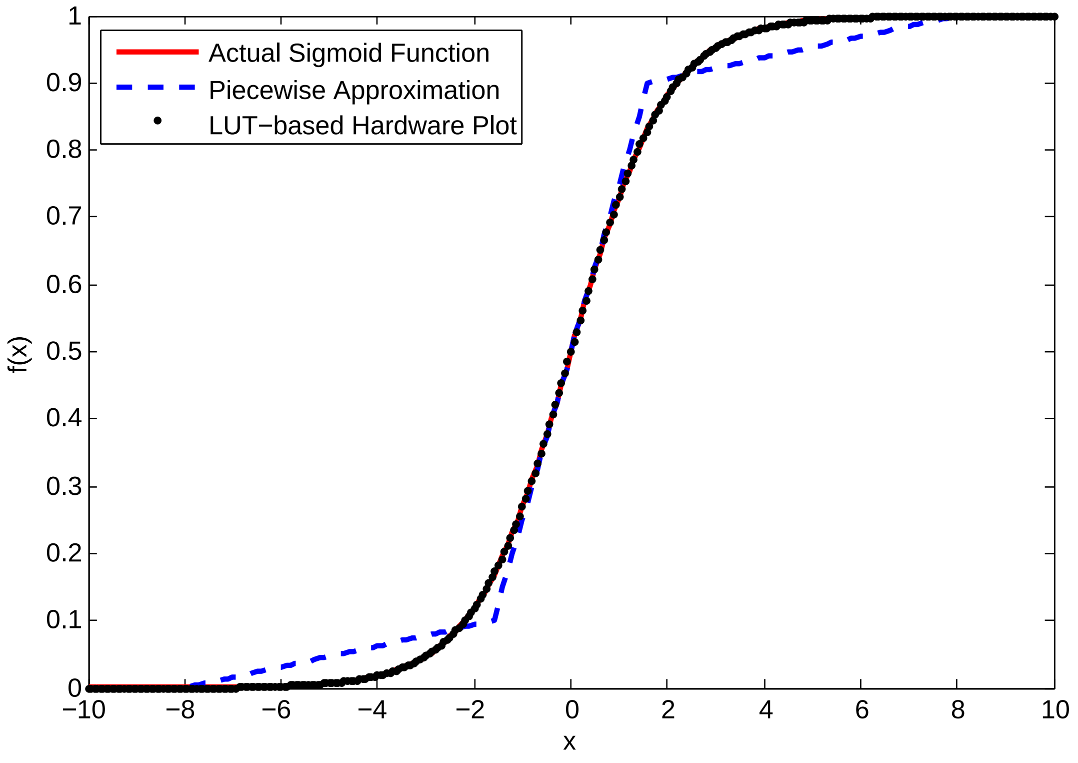


Figure 39: Comparison of Acutal Log-Sigmoid Function, Piecewise Linear Approximation and BRAM-Based Hardware Implementation

5.3.4 Performance Comparisons of Activation Functions

In order to provide a comprehensive perspective on all aforementioned activation function approximation methods, the results given by the actual continuous sigmoid function, PWL approximation approach and BRAM-based approach are all plotted in Figure 39. Besides that, each design is synthesized in Xilinx ISE 9.1.03i targeting the device of Virtex-5 XC5VLX50T-1ff1136. The synthesis results are compared in Table 8.

It is shown that the PWL Approximation approach is able to save 50% logic cells and 60% slice LUTs comparing with the LUT-Based scheme. Even attractive result is the timing constraint on critical path, where PWL Approximation is able to meet significantly more strict timing demands which make it more suitable for the high-speed design. Although

Table 8: Comparison of Synthesis Results for LUT-based, PWL Approximation, and Block RAM-based Approaches

Components/Modules	**LUT-Based**	**PWL**	**BRAM-Based**	**Available**
Number of Logic Cells[a]	166	84	0	46,080
Number of Slice LUTs	158	58	0	28,800
Number of bonded IOBs	26	28	0	480
Number of BUFGs	0	1	0	32
18K BRAMs[b]	0	0	1	120
Timing Constraints	22.73ns	3.228ns	—	—

[a] A single Virtex-5 CLB comprises two slices, with each containing four 6-input LUTs and four Flip-Flops. [301]
[b] Block RAMs are fundamentally 36 Kbits in size. Each block can also be used as two independent 18-Kbit blocks. [301]

the PWL approach has already provided a promising and efficient solution for the hardware implementation of complicated activation functions, the proposed BRAM-based LUT-like approach further addresses the challenge of hardware resource consumption on FPGA by making use of its built-in Block RAMs. Moreover, the actual resource saving would be expected once a bunch of neurons and the whole network have been implemented. Considering the negligible routing cost and the even faster on-line data access, the proposed BRAM-based is advocated in the following FPGA-based fault-tolerant and reconfigurable neural network platform.

5.4 BIDIRECTIONAL TIME-MULTIPLEXED ANN

The conventional feedforward neural network architecture, which is usually trained using the back-propagation algorithm, can be divided into the sequential execution of three stages known as feed-forward, back-propagation, and weight updating. The feed-forward stage is responsible for taking input patterns and propagating them through the network assigning an activation to every neuron according to Equation 3.3 and 3.4:

$$H_k^{(s)} = \sum_{j=1}^{N_s-1} w_{jk}^{(s)} o_j^{(s-1)} + \theta_k^{(s)}$$

$$o_k^{(s)} = f(H_k^{(s)})$$

The back-propagation stage finds the output errors and then propagates them backward through the network in order to find errors for neurons contained in hidden layers (Equation 3.6).

$$\epsilon_k^{(s)} = \begin{cases} t_k - o_k^{(s)} & s = M \\ \sum_{j=1}^{N_s+1} w_{jk}^{(s+1)} \delta_j^{(s+1)} & s = 1, \ldots, M-1 \end{cases}$$

After every non-input neuron has been assigned an error value, the update stage begins operation. The update state uses activation and error values found by the previous two stages to calculate the amount by which weights should be changed (Equation 3.7 and 3.8) and updates all weights with these changes (Equation 3.9).

$$\delta_k^{(s)} = \epsilon_k^{(s)} f'(H_k^{(s)}) \qquad s = 1, \ldots, M$$

$$\Delta w_{jk}^{(s)} = \eta \delta_k^{(s)} o_j^{(s-1)} \qquad k = 1, \ldots, N_s; j = 1, \ldots, N_{s-1}$$

$$w_{jk}^{(s)}(n+1) = \beta w_{jk}^{(s)}(n) + \Delta w_{jk}^{(s)}(n)$$

The completion of the update stage marks the end of the evaluation of one training pattern. This process is then repeated for all training patterns until the network is sufficiently trained.

Since the efficient resource utilization of FPGAs is always a great concern for hardware designers, and the neural network implementation is particularly resource-consuming due to its inherently complicated connections and massively computational demands, we propose and implement a *Bidirectional Time-Multiplexed Reusable ANN (BRANN)* architecture to maximize the system's resource utilization by reusing neuron units for both the hidden layer and the output layer. The involved neuron units will behave as either hidden neurons or output neurons in each of three stages in the back-propagation learning process. Besides the considerable reduction of resource consumption, another remarkable benefit of the proposed BRANN architecture is to reduce the autonomous recovery efforts reacting to any fatal error that destroys or disables some certain neuron units. Specifically, reusing the sensitive neuron units can firstly decrease the probability that the ANN system is out of order due to only one single "damaged" neuron in either hidden layer or output layer. Furthermore, reusing the neuron units can significantly reduce the design complexity of an autonomously reconfigurable ANN system, since one of the most challenging issues for the reconfigurability of ANN systems is the huge design space (i.e., a tremendous amount of different faulty cases for a large group of independent neuron units — one or more faulty neurons in hidden layer and one or more faulty neurons in output layer) that needs to be carefully considered so that the ANN system can adapt and reconfigure itself to react to any topology/behavior adjustments. In the current case, since the neuron units are reused for both layers, we only need to consider and treat the individual faulty neurons in a reduced "neuron pool" instead of a large set of neuron units. Given all design considerations presented above, the proposed BRANN architecture combines all hidden layers and the output layer together, as well as determines appropriate operations in each neuron node and the overall system behavior by using an ANN topological & algorithmic controller, which defines the architectural topology, inter-connectionism, layer composition, execution states, and intermediate training processes.

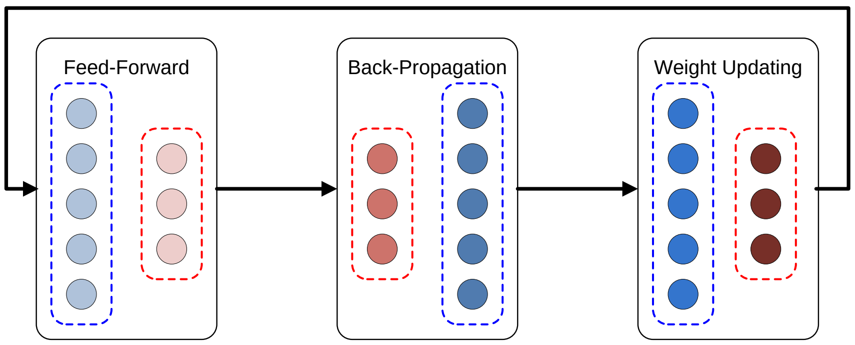
5.4.1 Time-Multiplexed Implementations

There are several possible options for implementing neural networks, as shown in Figure 40. The first option is probably the most intuitive way to implement a neural network, as shown in Figure 40(a). All involved computational neuron nodes are exhaustively implemented and deployed on hardware. For the three-stage training process, it seems like an extended structure with three cascaded neural networks, each of which is responsible for a certain type of operations out of the three training stages. Thus, it is manifest that such "flat design" will consume a tremendous amount of logic resources, although it can provide the best performance due to the most straightforward data path and potential pipelining capability.

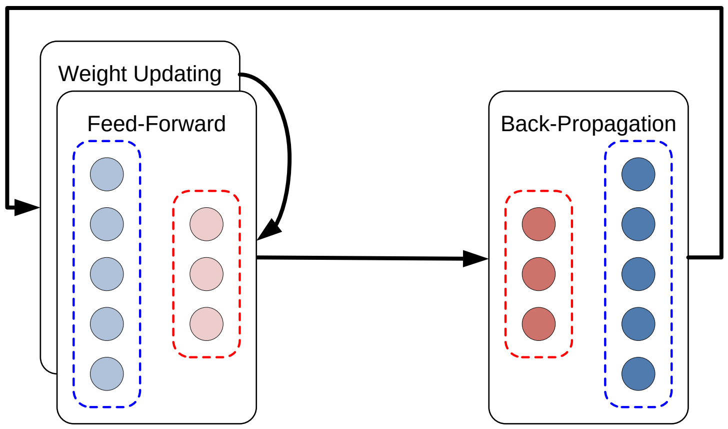
Since the feed-forward and weight updating stages have essentially the same data flow and operational sequence (except different operations on each neuron node). One possible improvement is to integrate these two processing stages together using the same hardware module in a time-multiplexed manner, as shown in Figure 40(b). Like the "Flat Design" option, the back-propagation stage still maintain its separate hardware module. Using this option, about 1/3 logic resource can be saved compared with the "Flat Design" option.

A more aggressive design option is to combine all three stages of operations into the same circuit module (Figure 40(c)), where various arithmetic operations need to be implemented in each neuron node in order to meet the requirements of all three computational stages. The specific stage and its associated operations are determined and managed by the ANN controller, according to work flow described in the back-propagation algorithms. With those enhanced multi-purpose computational neuron nodes and the sophisticated time-multiplexing control techniques, this design option only requires around 1/3 logic resource as the "Flat Design" option.

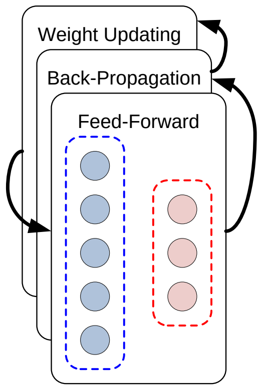
Although the design option presented above has already shrunk the ANN implementation to a large extent, we can present another further optimized neural network design with the highest degree of resource reuse. Observing that neuron nodes in either hidden layer or output layer have identical arithmetic functionalities besides their specific positions in the whole system operational flow, we propose to implement a generic neuron pool including a set of identical neuron nodes and involve some of them into the current operational stage according to training algorithmic flow, as shown in Figure 40(d). In this way, the best case is that the ANN system only needs a number of neuron nodes as many as the larger one out of the number of neurons required by the hidden layer and output layer. However, it can be easily imagined that, the control logics in this design option have to be carefully implemented to guarantee the correct timing sequences and data processing flows, and thus have the most design complexity out of all design schemes presented in this section. Since the target applications of our proposed ARANN system are usually power- and resource-concerned, we implemented the last option in this study and incorporated a set of independent, time-multiplexed, multi-purpose neuron nodes.

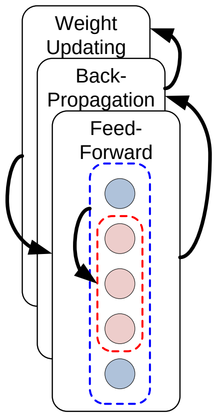


(a) Flat Design with Separate Neurons for Each Layer and Each Stage



(b) Two-Stages Reused Design with Separate Neurons for Each Layer





(c) Three-Stages Reused Design with Separate Neurons for Each Layer

(d) Three-Stages Reused Design with Shared Neurons for Different Layers

Figure 40: Design Options for Three-Stage MLP Neural Networks (Note: Blue circles — an array of hidden neurons; Red circles — an array of output neurons.)

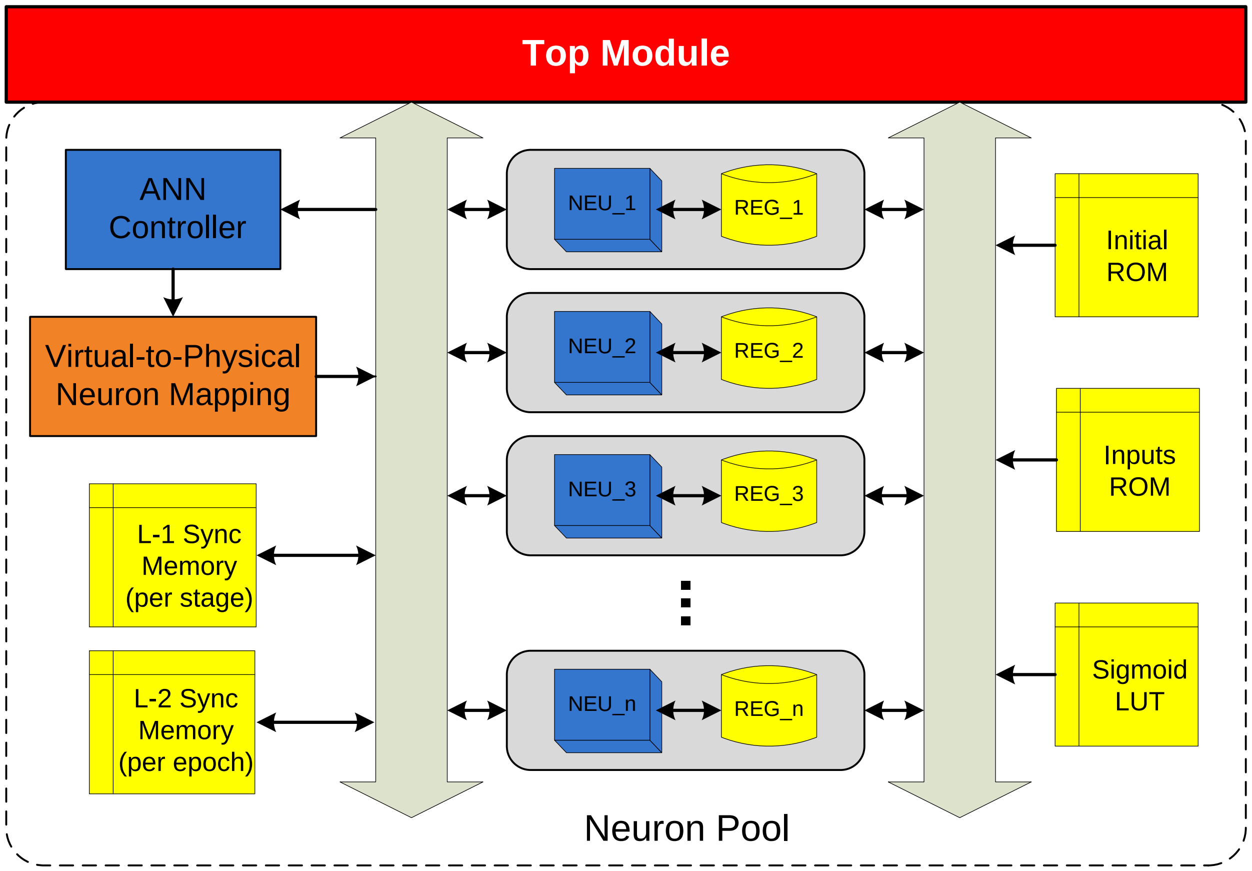


Figure 41: Architecture Diagram of The Implemented Multilayer Perceptron (MLP) Neural Network

5.4.2 Design Framework

Specially, the BRANN contains a top module enclosing a global controller, a Virtual-to-Physical neuron mapper, two synchronization memories, three ROMs (storing initialization information, input patterns, and activation function lookup table respectively), and a group of independent neuron units containing a separate arithmetic core and a private register (Figure 41). In what follows, we would like to present the detailed implementation solutions for each module within the BRANN.

5.4.2.1 ANN Controller The global controller is essentially an ANN topological & algorithmic controller, the key element within the neural network system which is mainly responsible for directing the work flow of whole system and sequencing the executions of local hardware subroutines on each neuron unit. As we discussed before, for a standard multilayer perceptron neural network, the learning and functioning processes involve considerable parallel computational efforts in many neuron nodes and thus introduce intricate data communications between all neuron nodes. This situation will be further complicated when we design a neuron-/synapse-reused neural network. The controller now has to explicitly describe the system work flow according to the back-propagation algorithm and assign appropriate neuron nodes to participate in the corresponding operations at each stage. A finite state diagram of the ANN system is designed within the controller, as shown in Figure 42. The basic processing steps include:

1. Initialization: to initialize all synapses (synaptic weights and biases), learning rate μ, and momentum factor β.
2. Input Loading: to load and present the training data (input patterns) to the network.
3. Feed-Forward Computation: to process received inputs in each neuron and propagate data from neurons in a former layer forward to neurons in the latter layer, according to the feed-forward connectionism.
4. Error Energy: to calculate the errors in the output layer — the difference between the expected target value and the actual neuron output value. If the ANN system is used to testing or predicting unknown patterns, then go directly to the Input Done step.
5. Back-Propagation Training: to calculate the local gradients according to obtained errors and the corresponding changes in synaptic weights (or biases) in backward manner until the first hidden layer.
6. Weight Updating: to update all the synaptic weights and biases based on the magnitudes of weight changes obtained in back-propagation training process.
7. Input Done: to determine whether there still are input patterns available or not. If other input patterns are still available, update the current neural network configuration information to the Level-1 Synchronization Memory and then go back to the Input Loading step to access another input pattern. Otherwise, it is indicated that the current training

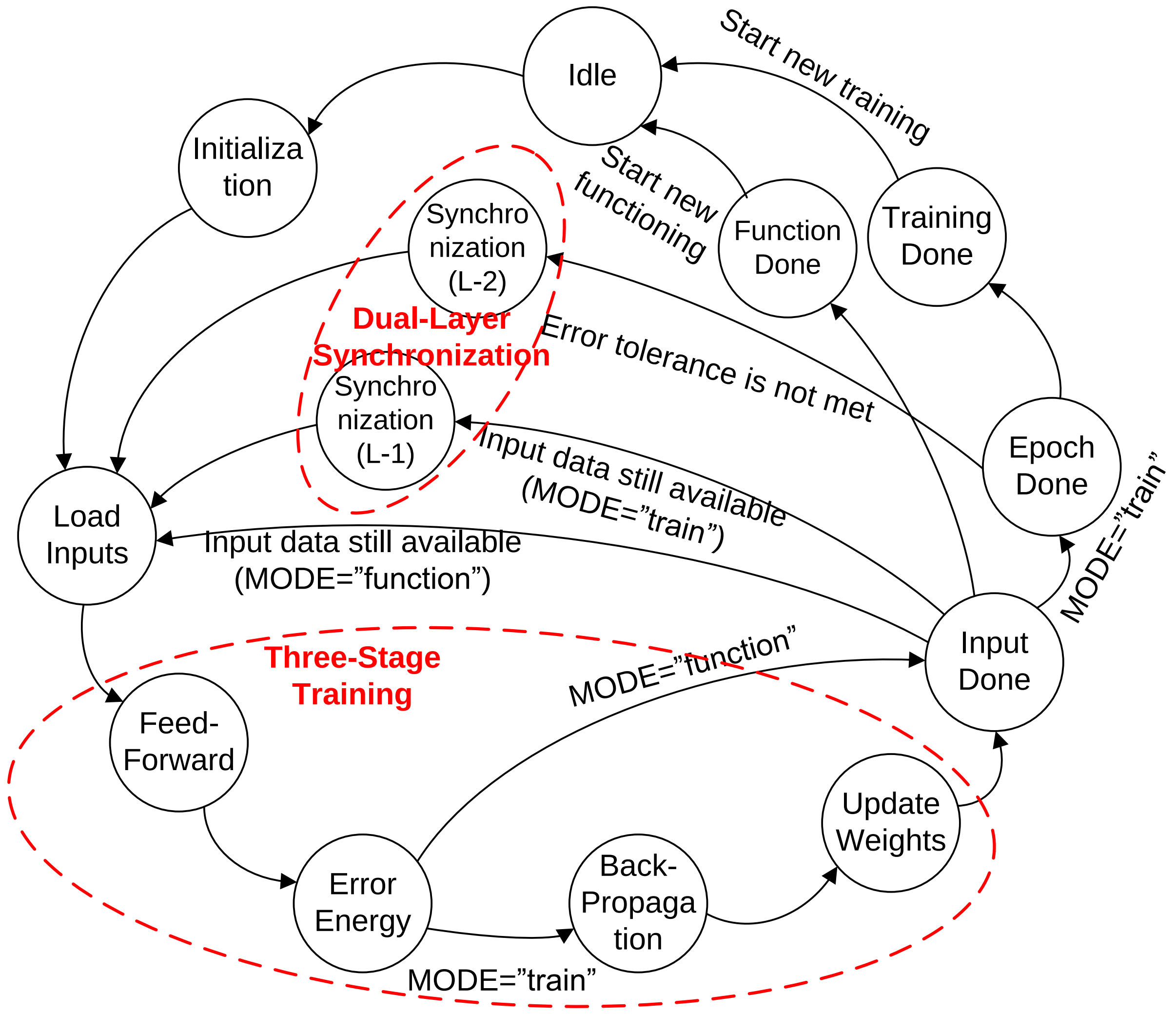


Figure 42: Finite State Diagram of The Implemented Multilayer Perceptron (MLP) Neural Network

or functioning tasks have been completed and the system will go into the Function Done state or Epoch Done state according to the specific operational mode of ANN system.

8. Epoch Done: to evaluate the accumulated error energy of the current neural network structure. For back-propagation learning, once all input patterns have been used to train the neural network, which is usually denoted as a training epoch, the ANN system has to check whether the current error energy can meet the error tolerance requirement and (possibly) start another training epoch from the Input Loading step until the error tolerance or some other stopping criteria is met.
9. Function Done: to indicate the successful accomplishment of the neural network's functioning process (i.e., prediction or classification). The system will go into the IDLE state waiting for new operational instructions.
10. Training Done: to indicate the successful accomplishment of the neural network's training process. The system will go into the IDLE state waiting for new operational instructions.

Given the state transition rules described above, the ANN Controller will monitor and direct the work flow for either training or functioning tasks of the neural network. More importantly, the ANN Controller will determine the specific neuron units that should be involved at a certain stage as well as the specific operations which involved neuron nodes should perform. However, the controller essentially does not care about the presence of one certain neuron, and what it is really concerned about is the number of neurons that can be used as specified in neural network model. On the other side, all neuron nodes perform certain operations according to the "instructions" assigned by the controller. Since all neuron units are functionally identical, thus they can be used at any location in the whole neural network. Decoupling the "virtual neurons" used in the ANN algorithm and the "physical neurons" implemented on hardware can significantly reduce the design complexity of the ANN Controller and increase the system flexibility.

Now, we have seen the design philosophy of such a *ANN Topological & Algorithmic Controller*. The block diagram of the controller as well as its major interfaces have been shown in Figure 43.

The global signals include CLK, RESET, START, READY, EPOCH, and RECOVERY STATUS. As the proposed ARANN architecture is capable of recovering ANN system perfor-

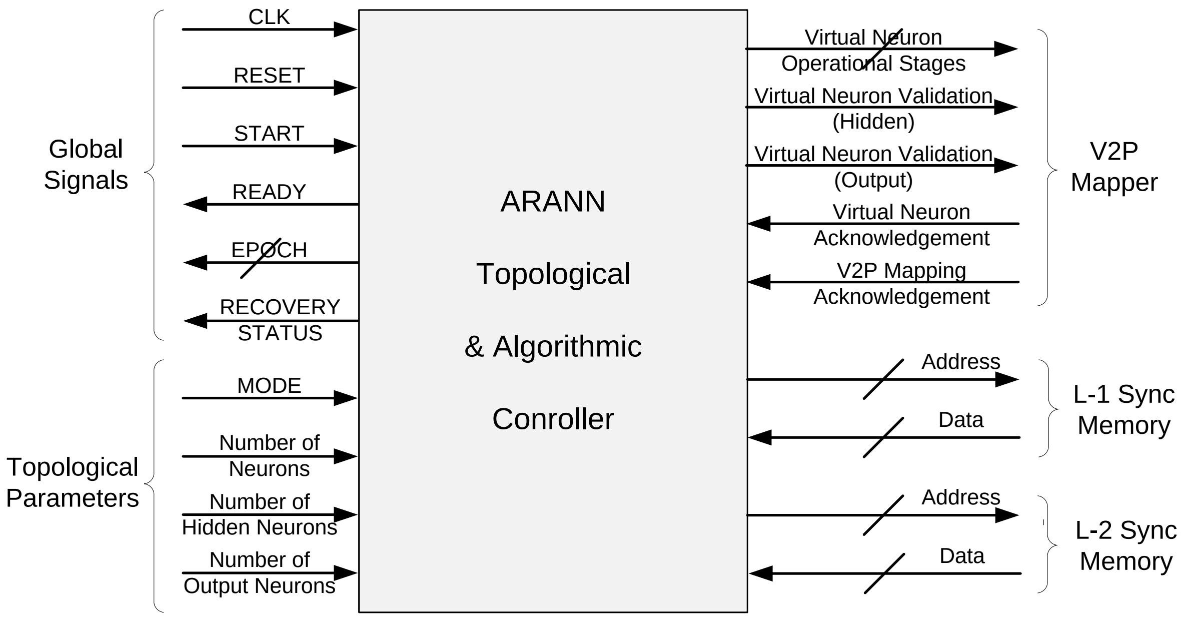


Figure 43: Block Diagram of the ANN Topological & Algorithmic Controller

mance by disconnecting faulty neurons and swapping space neurons into the neural network in response to unexpected faults, the RECOVERY STATUS signal is used to reflect the current system self-healing effort and to indicate whether the system can be completely recovered or not (due to the lack of available spare neurons). While one goal of this research is to build a fault-tolerant system and help it still maintain appropriate operational level even there are some damages in system, the system also has the capability to notify users if the system goes into a "compromised" operational mode (i.e., with less computational nodes in the neural network system).

The ANN topological parameters define all variables necessary to construct a neural network, including the MODE (i.e., "training" or "functioning" modes) as well as the number of all available neurons and the respective numbers of desired neurons in hidden layer and output layer.

Another major set of interfaces are all neuron controlling signals that are communicated with the Virtual-to-Physical Neuron Mapper. Basically, the specific involvement of each physical neuron unit is transparent to the controller and thus the controller only provides very simple controlling signals to direct the system execution. The *Virtual Neuron Validation (Hidden/Output)* signals are supposed to activate the neurons in hidden layer and output layer respectively, while the controller actually has no idea about the specific "physical neurons" that will be used in hidden layer or output layer at this moment. The *Virtual Neuron Operational Stages* signal specifies the desired operations on those activated neuron nodes. The acknowledgement signals indicate the accomplishments of calculations on the neuron and V2P neuron mapping respectively.

In addition, the controller also has the communication posts connected with two synchronization memories to guarantee the accurate system recovery (section 4.3.3) and optimize ANN's structure by comparing the performance of two similar network structures taking the network complexity into consideration (section 4.4).

5.4.2.2 Virtual-to-Physical Neuron Mapper In section 4.3.2, we have presented a *Decoupled Virtual-to-Physical Neuron Mapping* strategy, which is a critical element to achieve a more flexible, adaptable, and reconfigurable neural network system. Specifically, we propose a "neuron virtualization" by abstracting away the direct connections between ANN controller and all physical neurons, and inserting an adaptable V2P mapping block to determine an appropriate connections between virtual and physical neuron ports, according to the desired number of neurons and the availability of individual physical neurons. With such a decoupling scheme, the real spatio-temporal connections of "physical neurons" is transparent to the controller that handles "virtual neurons".

More specifically, the neurons appearing in the controller are essentially so-called "neuron symbols" or "neuron indexes". That is, any functionally correct neuron units can fill in these positions. Thus, we give a name to these neurons in the controller — "Virtual Neurons". Contrarily, the neuron units physically deployed on chip are named "Physical Neurons". In order to activate and manage some of physical neurons, the instructions that originally assigned to virtual neurons in the controller need to be transferred to the real physical neurons through one possible virtual-to-physical mapping scheme. This V2P mapping block can flexibly assign the virtual neuron indexes to any physical neuron ports, according to the desired number of neurons and the availability of each physical neuron. Once a physical neuron acquires the assignment of a specific virtual neuron index, it will be activated and used in the corresponding location of the neural network as indicated in the training/functioning algorithms.

The proposed *Decoupled Virtual-to-Physical Neuron Mapping* strategy has successfully addressed the reconfigurability and adaptability issues of conventional neural network implementations. It provides a convenient way to achieve the resource-efficient neuron reuse. More importantly, it indicates the possibility of increasing ANN's reliability by automatically reconfiguring and revising its structure in case one or more physical neurons are damaged. According to the thorough discussion and analysis in section 4.5, we primarily focuses on a *Cache-Accelerated V2P Mapper* in this study. Figure 44 illustrates the finite state diagram of the V2P Mapper. Starting from the IDLE state, the V2P mapper will transit to the V2P Allocation state and establish an appropriate mapping scheme between virtual neurons and physical neurons. The main constraint here is the availability of each physical neuron node, which is represented by a binary string where "1" indicates a faulty status and "0" indicates the normal case. The V2P mapper will adaptively search available physical neurons and assign them with certain virtual neuron indexes, until either the number of virtual neurons required by the controller has been met or all available physical neurons have been used (i.e., "compromised" operation mode). Once the allocation process is finished, the V2P mapper will move to the Synchronization state, where all assigned virtual neuron indexes will be sent out to each corresponding physical neuron node simultaneously. Moreover, the current mapping scheme will be stored into V2P Mapping Cache, which maintains all recently generated mapping schemes. After that, the V2P mapper will go to the Hold state and maintain all assignments on physical neuron ports, until there is another new request for the topology adaptation that may be caused by either hidden & output layer switches desired by ANN training/functioning algorithms or a newly detected faulty neuron node. It is worth mentioning that, since we have incorporated a cache into the V2P mapper, it no longer needs to go through the V2P allocation process every time. When a new topology is needed, the V2P mapper will first check the cache and access the corresponding mapping solution if there is a match between the new neuron availability condition and the stored one. Otherwise, it will initiate another allocation effort to establish a valid V2P mapping

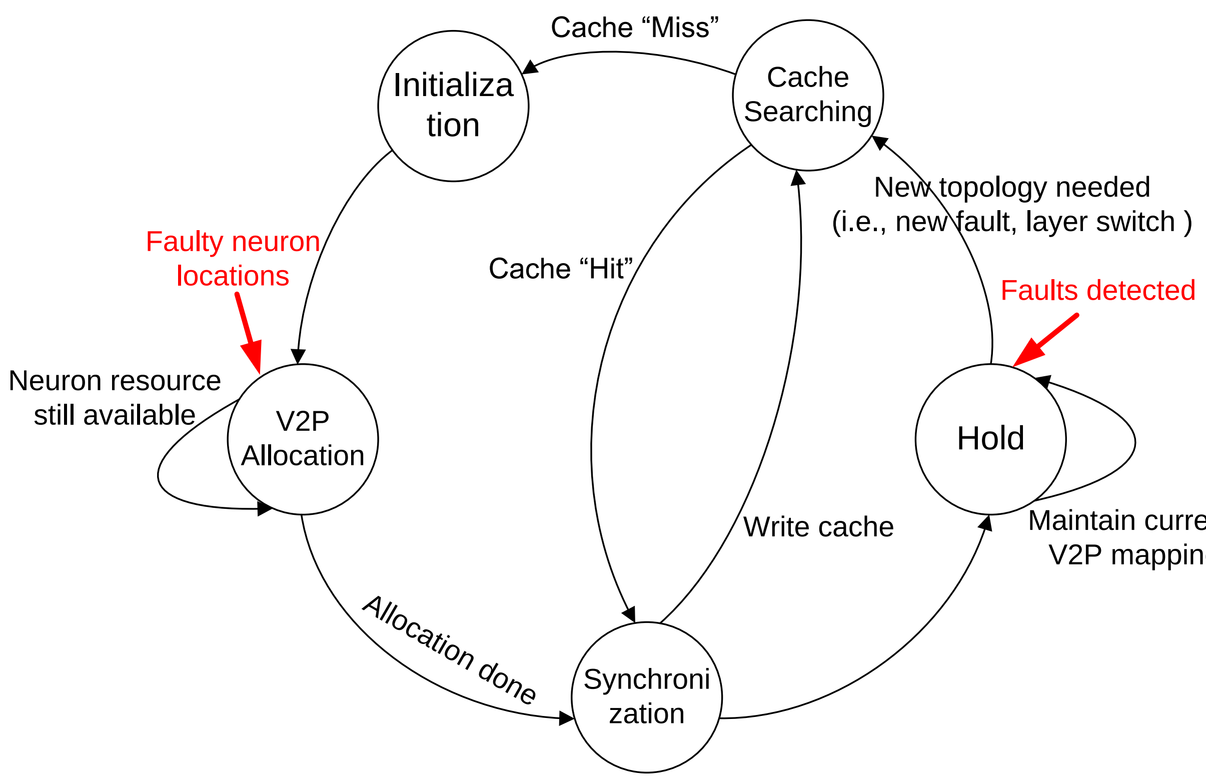


Figure 44: Finite State Diagram of The Virtual-to-Physical Neuron Mapping Block

scheme. It is shown that such cache-enabled strategy can significantly reduce the needs to perform extensive V2P allocation and accordingly reduce the time overhead, based on the characteristics of time locality existing for ANN topology adaptations.

Figure 45 shows the implementation block diagram of the proposed ARANN V2P Neuron Mapper. Similar as the ANN Controller, the V2P mapper also contains the global signals (i.e., CLK, RESET) and topological parameters (i.e., number of available neurons, hidden neurons, and output neurons). Besides that, V2P mapper has one unique external signal, named "Availability of Neuron", which is essentially represented by a bit-wise binary string where "1" indicates a faulty neuron and "0" indicates a normal neuron (All bits initially would be set to zero). The availability of each physical neuron will be reflected by the corresponding bit in this binary string, which may be determined and generated from an external error detector.

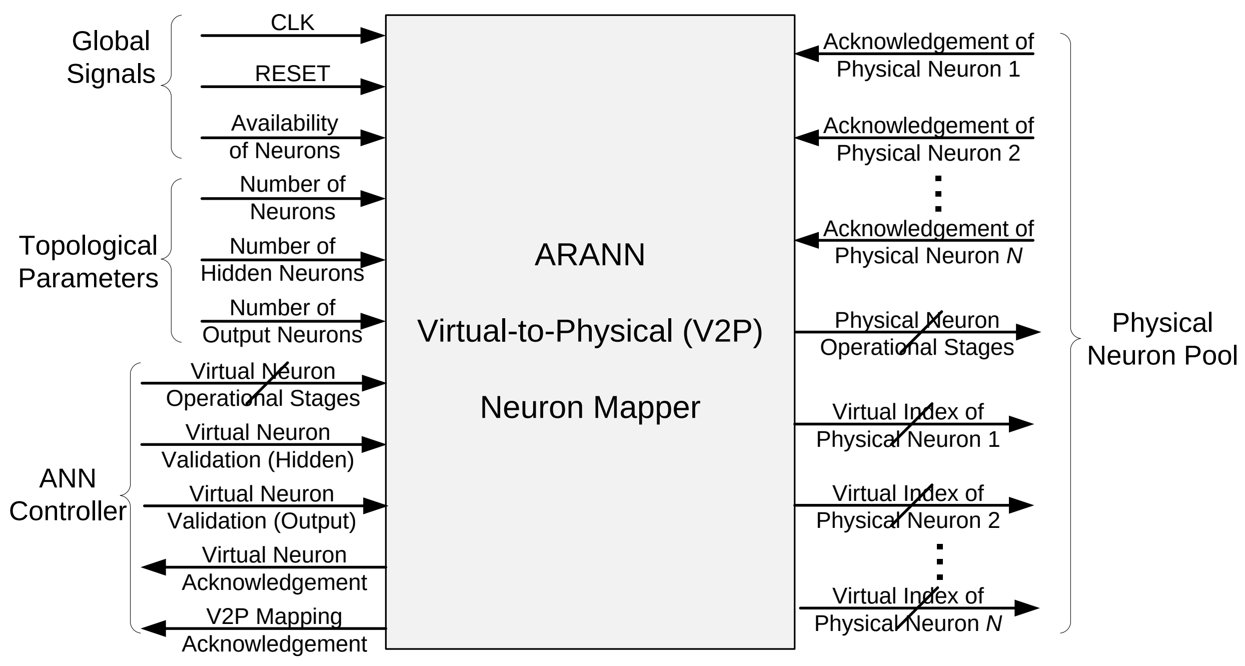

Global Signals
CLK
RESET
Availability of Neurons
Topological Parameters
Number of Neurons
Number of Hidden Neurons
Number of Output Neurons
ANN Controller
Virtual Neuron Operational Stages
Virtual Neuron Validation (Hidden)
Virtual Neuron Validation (Output)
Virtual Neuron Acknowledgement
V2P Mapping Acknowledgement
ARANN
Virtual-to-Physical (V2P)
Neuron Mapper
Acknowledgement of Physical Neuron 1
Acknowledgement of Physical Neuron 2
Acknowledgement of Physical Neuron N
Physical Neuron Operational Stages
Virtual Index of Physical Neuron 1
Virtual Index of Physical Neuron 2
Virtual Index of Physical Neuron N
Physical Neuron Pool

Figure 45: Block Diagram of the Virtual-to-Physical Neuron Mapper

Since the ANN Controller only determines a certain group of neurons that should be activated and involved at a certain stage, as well as the specific operations which involved neuron nodes should perform, the V2P Mapper is supposed to appropriately transfer such activation signals and operation instructions to some physical neurons by establishing a virtual-to-physical neuron mapping scheme. For instance, once the V2P Mapper receives the Hidden Validation signal, which means all "hidden neurons" should be activated at this moment even though the controller does not know what the specific hidden neurons are, it needs to adaptively search available physical neurons and assign them with virtual neuron indexes. A non-zero index will be able to activate the physical neuron. Correspondingly, the operational instructions will be forwarded to those activated physical neuron nodes for directing them to perform some desired operations. The V2P Mapper has the interfaces to each physical neuron node, which basically include a virtual neuron index that is assigned according to the availability of physical neurons and the number of neurons specified by the controller, an operational stage signal indicating the desired operations on those activated neuron nodes, and an acknowledgment signal representing the accomplishment of desired operations on the neuron node.

5.4.2.3 Neuron Units Neurons are the fundamental computational units responsible for performing all computations needed for the feed-forward functioning or back-propagation training of neural networks. As shown in Figure 46, each neuron unit is primarily made up of a neuron arithmetic core, a register file, and some control logics. Since the neuron is mainly designed to achieve a variety of operations, each neuron has a bunch of communication interfaces with all external memory elements, such as the Initialization ROM, the Input ROM, the Lookup Table ROM of Sigmoid Activation Functions, and two Synchronization Memories. Given the virtual neuron index assigned by the V2P Mapper and the operational instructions provided by the ANN Controller, the control logics within the neuron will start to perform appropriate operations on the arithmetic core, using the data accessed from its internal register file or from other external memories.

The arithmetic core within the neuron is one of the most critical components and plays a significant role in the overall functionality of neuron network. For the ARANN implementation, a fast, resource-efficient, and low-power neuron arithmetic core is highly desired in the system design. In this study, we have identified six main operations for each neuron (i.e., $A + B$, $A \times B$, $(A - B)^2$, $(A - B) \times B$, $A \times (1 - B)$, and $\sum A_i \times B_i$) and have successfully implemented a highly efficient neuron core using the embedded XtremeDSP-48E slices in FPGAs (Please refer to the design details in Section 5.2). According to the operation mode specified by the *Op_mode* signal (i.e., one out of six arithmetic operations), the neuron core will perform corresponding arithmetic operations using two operands — *Op_a* and *Op_b*, and generate the final results in the *Output* port. It is worth mentioning that, in order to achieve the highest performance and largest throughput, the neuron core is dedicatedly designed to feed one pair of operands every cycle and send intermediate results back to the input ports of the accumulation adder in a fine-grained pipelining manner (as shown in Figure 38). Therefore, the input signal *Op_count* is used to specify the total number of operand pairs that are fed into the neuron core continuously.

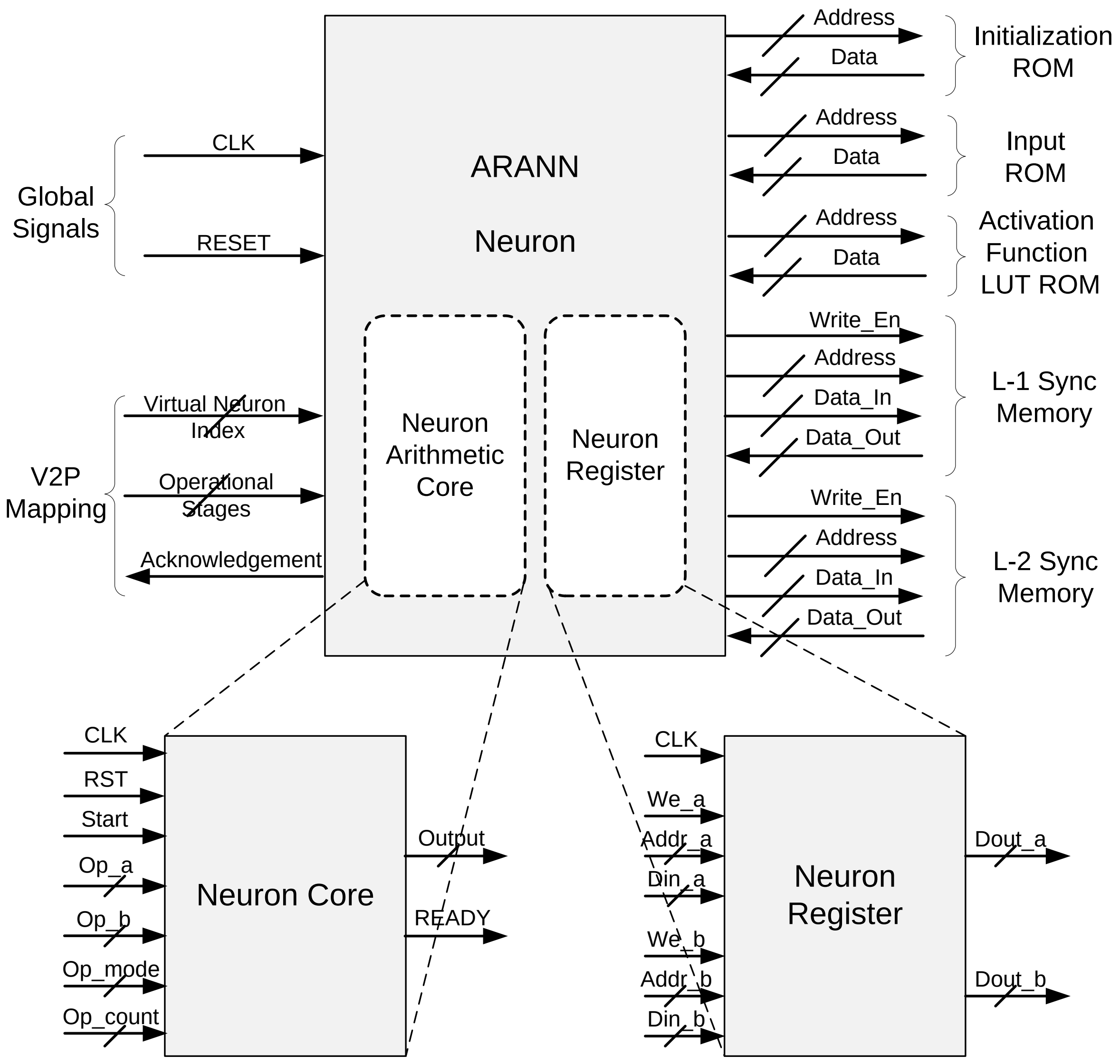


Figure 46: Block Diagram of the Physical Neuron Unit in ARANN

To alleviate the performance burden caused by a tremendous amount of data communications between the controller, all neuron units and memories, we proposed to integrate a private register file into the neuron unit and thus move most of data flows into a local region. Such private registers are dedicatedly used to store synaptic information (e.g., inputs, targets, weights, biases) and to buffer error values as a scratch pad. Neuron registers are implemented as dual-port read and one-port write RAM block, with a width of 16 bits and a depth of 512 (Figure 46). The detailed contents in the register are illustrated in Figure 47. It is shown that the majority of intermediate results and neural network configurations are stored in this private register and can be easily accessed during the operations.

5.4.2.4 Memory Units Besides the controller and basic neuron units, we need several other memory units to store information needed by the ANN system. For instance, as we described above, an initialization ROM is implemented to initialize all synapses and systematic parameters. Another input/target ROM is designed to store all input patterns and corresponding target values used for the training process of the ANN system. Once the ANN architecture has been established and well trained, all remaining studying tasks will be executed using the input data directly sent to the ANN system in a real-time manner. Moreover, as we discussed in section 5.3, considering a balanced tradeoff between system accuracy and resource consumption, a Lookup Table for the Sigmoid Activation Function was implemented in our experimental testbed. Consuming significantly less logic resource, this lookup table can provide comparable accuracy as an exhaustive hardware implementation of the sigmoid functions (as shown in Figure 39). Besides all aforementioned memories, another Output memory is optional, which can be used to temporarily store all generated outputs from the ANN system and their errors comparing to the expected target values. The user can easily switch to directly send out all generated outputs via FPGA I/O ports.

5.4.3 Three-Stage Learning Process

Given the proposed bidirectional time-multiplexed neural network design scheme and the presented implementation details of all major components in ARANN, we would like to

Name	Address	Content
RegIHB	0	Bias (Hidden Layer)
RegIHW	1	Weights (I to H)
RegHOB	RegIHW+NumInt	Bias (Output Layer)
RegHOW	RegHOB+1	Weights (H to O)
RegInt	RegHOW+NumHdn	Inputs
RegTgt	RegInt+NumInt	Targets
RegHOut	RegTgt+NumOut	Hidden Outputs
RegOut	RegHOut+NumHdn	Outputs
RegErr	RegOut+NumOut	Errors
RegErrAll	RegErr+NumOut	Total Error Energy
RegDelO	RegErrAll+1	Delta Outputs
RegDelH	RegDelO+NumOut	Delta Hidden Outputs
RegDelHW	RegDelH+NumHdn	Delta Weights (I to H)
RegDelOW	RegDelHW+NumInt	Delta Weights (H to O)
RegAlpha	RegDelOW+Hdn	Learning Rate α
RegBeta	RegAlpha+1	Momentum Factor β
RegLrnIH	RegBeta+1	Learning Terms (I to H)
RegLrnHO	RegLrnIH+NumInt	Learning Terms (H to O)
RegTemp	RegLrnHO+NumHdn	Intermediate Values

Figure 47: Register Organization for Each Neurons (Note: **NumInt** — Number of inputs; **NumHdn** — Number of neurons in the hidden layer; **NumOut** — Number of neurons in the output layer.)

review the three-stage back-propagation learning process again to demonstrate the specific operational control sequences and data flows on all aforementioned processing elements.

1. ***The Feed-Forward Stage***

   To begin the feed-forward stage, the ANN controller firstly initiate a hidden neuron activation signal and a corresponding operational instruction signal, both of which will be translated by the V2P Mapper into the virtual neuron indexes that then will be forwarded to relevant physical neuron units. A non-zero virtual neuron index is able to activate the physical neuron node and indicate its relative location within the hidden layer or output layer. As the activation of a certain neuron node, the operational instruction signal will direct the activated neurons to perform appropriate operations. At this moment, all involved neurons (in hidden layer) will access the Input ROM to obtain the current input pattern, which will be also stored into the private register within the neuron node, and then calculate the pre-synaptic value (i.e., weighted sum of inputs). Using the pre-synaptic value as the indexing address, the neuron node will access the Lookup Table of nonlinear Sigmoid activation function and obtain the post-synaptic output, also called "hidden output" (Equation 3.3). Each hidden neuron will send their generated hidden output to an external memory and synchronize it with all other neuron nodes to ensure every neuron will have all output information generated from hidden neurons.

   Once the operations on all involved hidden neurons are accomplished, the ANN Controller takes over the system control again and send out an activation signal to "turn on" the neurons that can be used in the output layer. Similarly, the V2P Mapper will determine an appropriate mapping scheme and distribute the valid virtual neuron indexes to a set of involved physical neuron nodes. Those newly activated nodes will behave as the output neurons and perform the same operations using the hidden outputs generated in last step (Equation 3.4). Once the outputs are generated, they will be synchronized with other neurons and used to calculate the error energy. The feed-forward stage ends by finding the difference between the target outputs and the network's outputs.

2. ***The Back-Propagation Stage***

   The back-propagation stage begins by finding the errors for the output layer using the differences found above according to Equation 3.6. Then, ARANN calculates the changing magnitudes of synaptic weights (Equation 3.8) on each output neuron in the network (including the summation) simultaneously. Once the weight changes for output neurons have been generated, the ANN Controller will again activate the desired hidden neurons and the back-propagation operations on them.

Unfortunately, the back-propagation operation on hidden layers consists of the most complicated data accesses and computations in the whole ANN training process. The main reason is the interleaved storage scheme of synaptic weights. Specifically, in current design, all neurons only keep the weight information associated with synaptic connections ending at themselves. For instance, the weight w_{ij}, which represents a synaptic connection from neuron i to neuron j, is stored in the private register of neuron j. Such storage mechanism can significantly facilitate the feed-forward process, which involves the calculation of the weighted sum of all inputs and synaptic weights connected to a neuron. However, in back-propagation process, the neuron has to calculate the error term using all synaptic connections starting at itself (Equation 3.6). For instance, neuron i now needs to access all associated weights w_{ij}. In this case, each hidden neuron has to access all of its subsequent neuron nodes respectively and this interleaved data access has to be performed sequentially, causing a major performance bottleneck in the ANN system. Each hidden neuron calculates the accumulated sum of error-weight products $\delta \cdot W$ and then multiply the error term with the appropriate activation derivative. This process is then repeated for every neuron in the hidden layer, until all potential changing magnitudes of synaptic weights have been determined (Equation 3.8).

3. ***The Weight Updating Stage***

The updating stage begins in a similar manner as the feed-forward stage with the global controller activating all neurons in the hidden layer. Based on the synaptic changing magnitudes ΔW determined by the first two stages, the neurons will immediately update its associated synaptic weights (Equation 3.9) and the new weight values are written back to memories to replace old weights. This process is repeated for every weight in the hidden layer in a parallel manner. Then, ARANN begins changing the weights between the hidden layer and the output layer (in a three layer network). The accomplishment of the updating stage indicates the end of one input training pattern.

These three stages are repeated for each input pattern in the training set (one epoch), which could last for a number of epochs until the network is sufficiently trained (i.e., a reasonably small error energy).

6.0 RESULTS AND ANALYSIS

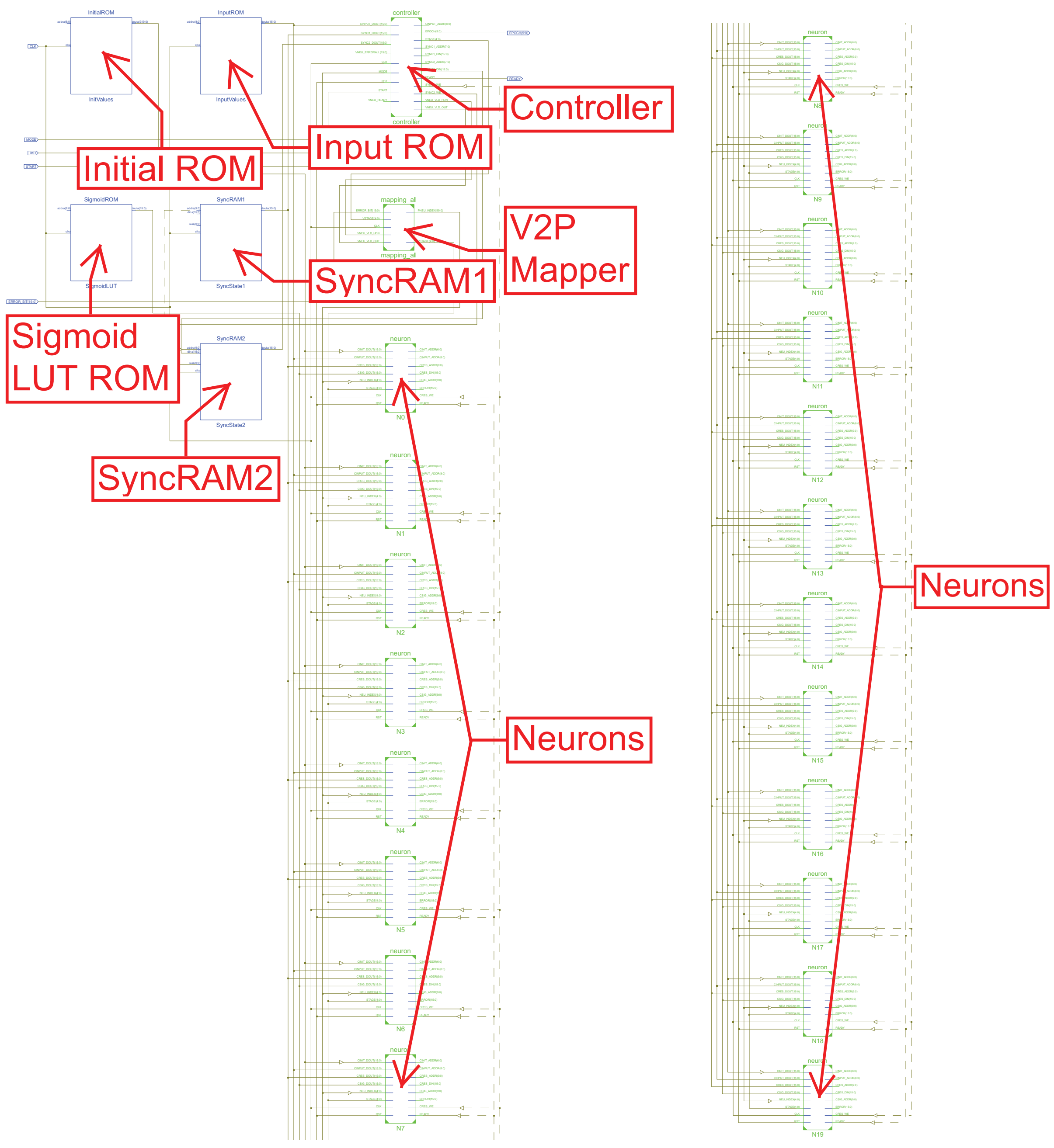
Experimental results are presented and discussed in this chapter. We first demonstrated the implementation details of the proposed ARANN architectural framework on hardware and the corresponding FPGA-based prototype. Following the implementation evaluation is the discussion on the effectiveness of ARANN framework in application level. Specifically, we evaluated the training performance of ARANN system using two examples: a simple classification case and a more complex biomedical application using neural network to model the intricate correlations between limb muscular activities and end-point locomotion behaviors (as we presented in section 3.3). Then we illustrated ARANN's self-healing process which reacts unexpected faults on neuron units and maintains appropriate operational level of system, by automatically adapting and reconfiguring the system structure and topology. Finally, we showed ARANN's self-optimizing capability of exploring a Pareto-optimal neural network structure for a given application on the fly, based on the hybrid system cost function measured by both classification accuracy and complexity overhead.

6.1 IMPLEMENTATION OF ARANN

The proposed ARANN architectural framework is designed and implemented with Verilog-HDL in a highly modularized way, including an ANN Topological & Algorithmic Controller module (refer to section 5.4.2.1), a reconfigurable Virtual-to-Physical Neuron Mapping module (refer to section 4.5.2 and 5.4.2.2), 20 homogeneous neuron modules including separate neuron arithmetic core and associated register file (refer to section 5.4.2.3), an Initialization ROM (used for initializing synaptic parameters), an Input ROM (used for providing training patterns), a Sigmoid Activation Function Lookup Table ROM (refer to section 5.3), and (optional) Output/Error ROMs (used for storing generated outputs and errors). According to the discussion in section 5.1, we use 16-bit fixed-point data format in the whole design. The ARANN design is physically synthesized, floorplanned, placed and routed on the Xilinx Virtex-5 XC5VLX110T FPGA using ISE 11.1i, PlanAhead 11.1i, and ModelSim SE 6.5 design tools. The final schematic diagram is shown in Figure 48.

Table 9 shows the resource consumptions and chip footprints of all major components (i.e., controller, V2P mapper, and neuron unit) and the whole ARANN system. The corresponding percentage rates of hardware resource utilization are also included in brackets. According to the XPower Analyzer [3] in Xilinx ISE 11.1, the total estimated power consumption of ARANN is around 1358.93mW.

As we discussed in section 4.3.1, in order to provide more systematic flexibility for neural networks, we proposed a *Distributed Artificial Neural Network (DANN)* implementation architecture. Instead of grouping a single overweight centralized controller and a set of basic computational nodes, DANN implements a group of independent, autonomic, smart neuron units containing their private arithmetic cores and register files, as well as deploys a lightweight controller which is only responsible for directing the ANN training/functioning processes according to certain algorithms. Such design strategy has been exactly reflected by their respective chip footprints in Table 9, where the controller takes up of 2% of the overall consumption of logic resources, in contrast to the portion of 4.5% for one individual neuron. As a representative *Centralized Artificial Neural Network (CANN)* architecture, Sun [268] presented a multilayer perceptron neural network implementation, which are primarily made up of one dedicatedly design global controller and 30 simple neuron nodes. In order to demonstrate the difference between these two distinct implementation strategies, we analyzed the footprint percentages of the major components in these two design examples, as shown in Figure 49. It is worth mentioning that those percentage numbers are estimated based on the respective synthesis results of individual component on Xilinx FPGAs. According to this figure, it is shown that the DANN shows a significantly unbalanced footprint distribution, where highly autonomous neuron units occupy the majority of neural network hardware. On the other side, as what we imagined, the complicated global controller and other elements



(a) Top Half of the Schematic

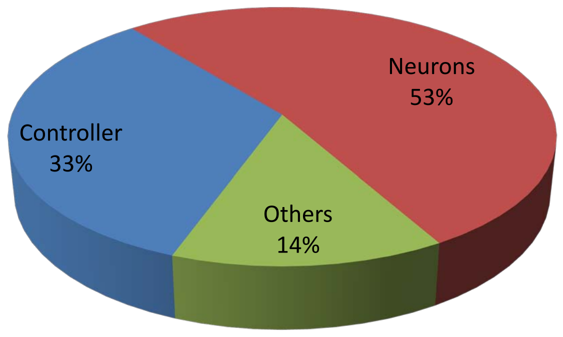
(b) Bottom Half of the Schematic

Figure 48: Schematic Diagram of the Implemented ARANN (20 Neurons)

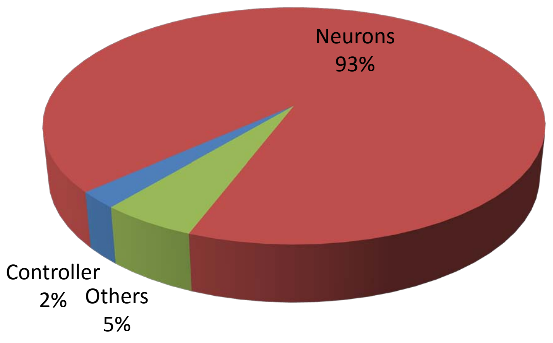
Table 9: Synthesis Results of ARANN Components

Properties	Controller	V2P Mapper	Neuron	ARANN	Available
Slice Registers	550(1%)	313(0%)	1097(1%)	22804(32%)	69120
Slice LUTs	1068(2%)	443(1%)	3068(4%)	66375(96%)	69120
Fully used LUT-FF pairs[a]	535(51%)	313(70%)	1086(33%)	22720(34%)	N/A
Bonded IOBs	116(18%)	133(20%)	146(22%)	35(5%)	640
BUFG/ BUFGCTRLS	1(3%)	1(3%)	1(3%)	2(6%)	32
DSP48Es	2(3%)	0	5(7%)	22(34%)	64

[a] The percentage rates of *fully used LUT-FF pairs* are based on the used resource of each individual component, instead of the whole system.



(a) Footprint Percentages in a Centralized ANN Design Example [268]



(b) Footprint Percentages in ARANN

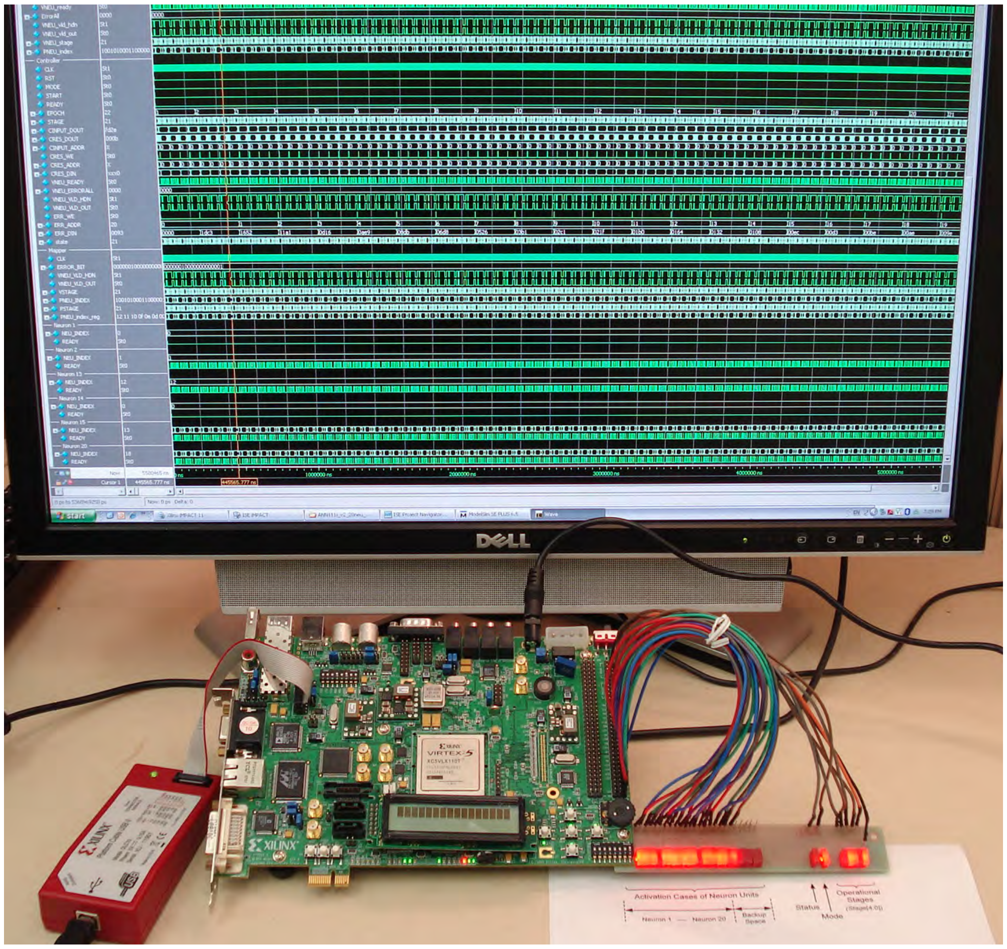
Figure 49: Footprint Percentages of Major Components in Centralized and Distributed ANN Design Examples

(memories, routing paths, interfaces, etc.) consume considerable logic resources in a neural network hardware.

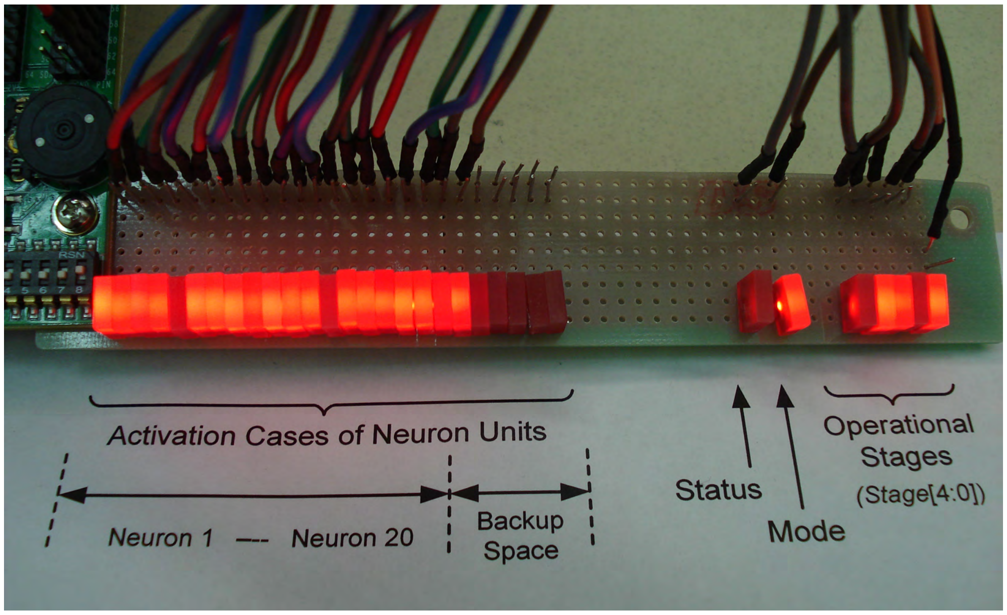
Comparing with CANN, the proposed DANN architecture has many remarkable advantages. Firstly, the highly autonomous neuron units can significantly improve the system performance by maximizing the degree of neuron-level parallelism. Secondly, DANN greatly alleviates the burden of data communications among controller, memories and neurons. Thirdly, incorporated with a flexible "neuron virtualization" strategy, DANN makes all neurons behave as independent processing elements and provides a reconfigurable infrastructure for the ANN structural adaptation and optimization. In this study, we particularly investigated biologically-inspired autonomous architectural reconfiguration approaches, which are able to recover the system when one or more neuron units are unexpectedly damaged, to address the reliability issues of neural network hardware. Unfortunately, the proposed ARANN is not applicable for the cases where faults occurring on other components besides the neuron nodes. More importantly, given the prominent footprint percentage of neuron units in ARANN, it would be very convenient and cost-effective to achieve a higher level full-system fault-tolerance, by replicating all other components (e.g., controller, memories, or other logics) in a conventional triple modular redundant manner. Contrarily, for a centralized neural network design, it is still very challenging to achieve a full-system fault tolerance and particularly expensive to replicate a large portion of the ANN system. Thus, the DANN architecture has superior properties and advantages, as far as the reliability and flexibility are concerned for an ANN system design.

Finally, the FPGA-based prototype of the proposed ARANN architectural framework is shown in Figure 50(a), where a Xilinx ML505 FPGA board and the external simulation environment have been presented. In order to better demonstrate the system structural adaptations on this ARANN prototype, we build a LED display panel, which is capable of displaying several major internal signals of ARANN and thus reflecting the system operational status. Specifically, as shown in Figure 50(b), the LED array primarily consists of four LED groups.

The first LED group, also the largest one, is made up of 25 LEDs (including 5 spare LEDs), each of which indicates if the corresponding neuron unit in ARANN is enabled (bright) or disabled (dark). Thus, the activation status of each individual neuron (out of the total 20 neurons implemented on this ARANN prototype) can be easily observed during



(a) FPGA-based ARANN Prototype and Simulation Environment


Activation Cases of Neuron Units
Neuron 1 --- Neuron 20
Backup Space
Status
Mode
Operational Stages (Stage[4:0])

(b) Close-Up View of LED Display Panel

Figure 50: The FPGA Prototype of the Proposed ARANN System

the system operations. Accordingly, in case there is any structural adaptation in ARANN, the dynamically reconfigured neural network structure involving different set of neuron units could be reflected by these LEDs.

The second group, consisting of two LEDs, indicates the specific operational status or mode of ARANN. As we discussed before, the proposed ARANN can react to the faults occurring on any neuron by autonomously reconfiguring its structure. The faulty neuron will be disconnected from the main neural network and another available neuron unit could be swapped into the network. However, if there is no spare neuron available, ARANN can still perform its regular operations with less neuron units involved, which we call the "compromised" operational status. The STATUS LED indicates if the ARANN is currently running in a normal status (dark) or a "compromised" state (bright). The other MODE LED represents the current functioning mode of ARANN, that is, either training mode (bright) or classification mode (dark).

The remaining 5 LEDs, belonging to the third group, show the operational stages specified by the ANN Controller. Such stage information is used to direct the specific operations in each neuron node, like a certain type of computer instructions. Basically, we implemented 30 different operations in each highly independent "smart" neuron, for the whole feed-forward back-propagation training process. According to the operational instructions provided by the controller, all involved neuron nodes will enter a particular stage during the whole training/functioning process of neural networks and conduct some sort of operations simultaneously.

6.2 TRAINING PERFORMANCE OF ARANN

As we presented before, artificial neural networks (ANNs) have evolved into a big family with many algorithmic variants and have been successfully applied to a variety of biomedical applications since 1980s. Among all of ANN's functionalities, two most important and widely identified tasks are *pattern recognition* (also known as classification) and *regression* (also known as function approximation). The former aims to classify data (patterns) based on *a priori* knowledge, which can be usually groups of measurements or observations. While the latter focuses on exploring the underlying relationship between one or more dependent variables and independent variables, and thus establishing a mathematical model to describe such relationship. More specifically, function approximation helps us understand how the dependent variables change when any of independent variables are varied. In order to testify and verify the efficacy and efficiency of the implemented ARANN system, we apply ARANN to two real biomedical cases in this section.

Cardiovascular disease (CVD) is caused by disorders of the heart and blood vessels and by far continues to be the leading cause of death in the world. CVD encompasses a variety of cardiac conditions including heart attack and hypertension. According to the American Heart Association, in the United States alone 81,100,000 people are estimated to have one or more forms of CVD and nearly 2,300 Americans die of CVD each day [174]. Cardiac arrhythmia, defined as abnormal heart rhythms, is a very common type of CVD and is thought to be responsible for most of the sudden cardiac deaths that occur every year. The most common test for a cardiac arrhythmia is an electrocardiogram (ECG), which measures the electrical impulses of the heart via electrodes on the skin's surface. Over the past several centuries, many researchers have established various effective approaches to identify ECG morphological features and detect ECG abnormalities associated with one or more CVD conditions based neural network-based algorithms [47, 120, 119, 134, 208, 261]. Such ANN-based solutions are essentially to achieve pattern recognition by classifying the ECG patterns (representing periodical heart beats) into different heart beat types indicating possible level of arrhythmia.

In an effort to facilitate future pervasive healthcare, we previously presented a mobile platform — *HeartToGo* — capable of monitoring and recording ECG in real time, performing continuous on-line ECG processing , automatically detecting and classifying abnormal CVD conditions [135]. The experiments based on MIT-BIH Arrhythmia Database [189] shows roughly more than 90% classification accuracy when classifying 5,421 QRS complex templates into five classes.

In this study, we use the same experimental data to testify the implemented ARANN system. Consistent with the neural network model presented in *HeartToGo* [135], we adapt

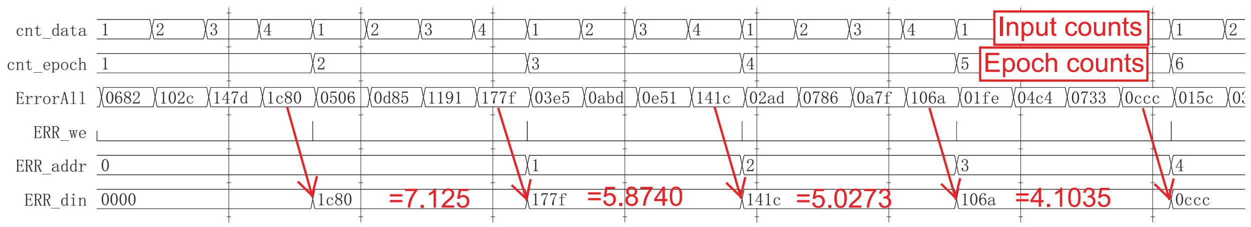


Figure 51: Waveform Segment of Training Process in Post-Synthesis Simulation

the ARANN to contain 51 inputs and 12 outputs, where 51 inputs represent the data samples selected out of each heart beat and 12 outputs represent the totally twelve possible beat types. To provide a much more clear view on the ARANN's training process, we only choose 4 heartbeats as training patterns, which contains 1 normal heartbeat and 3 abnormal ones (PVC — Premature Ventricular Contraction, PACE — Paced Beat, and RBBB — Right Bundle Branch Block Beat). The heartbeats all selected from patients' records in MIT-BIH database: the normal one (No. 100 record), the PVC (No. 119 record), the PACE (No. 104 record), and the RBBB (No. 118 record). Following the back-propagation learning procedure described in section 3.2, these four training patterns will be periodically fed to the neural network until the accumulated error energy achieves a reasonable low level or other specified termination criterion is met.

A piece of simulation waveform is illustrated in Figure 51, where the overall error ("ERR_din") has decreased from 7.125 $(1C80)_{16}$ to 4.1035 $(106A)_{16}$ when the training process continues to the 5th epoch ("cnt_epoch"). It is also shown that the four input patterns ("cnt_data") that constitutes each training epoch and their corresponding accumulated error items ("ErrorAll"). The decrease trend of the overall error is also applicable to each input pattern, for example, the error of $(0682)_{16} \rightarrow (0506)_{16} \rightarrow (03E5)_{16} \rightarrow (02AD)_{16} \rightarrow \cdots$ for the 1st input.

The decreasing trend of the total error energy is further demonstrated from Figure 52, where the error has reduced to a significantly small level after 40 training epochs. Based on this well-trained ANN system, the training efficacy and accuracy are testified using another group of input patterns presented to the implemented ANN platform. The average correct

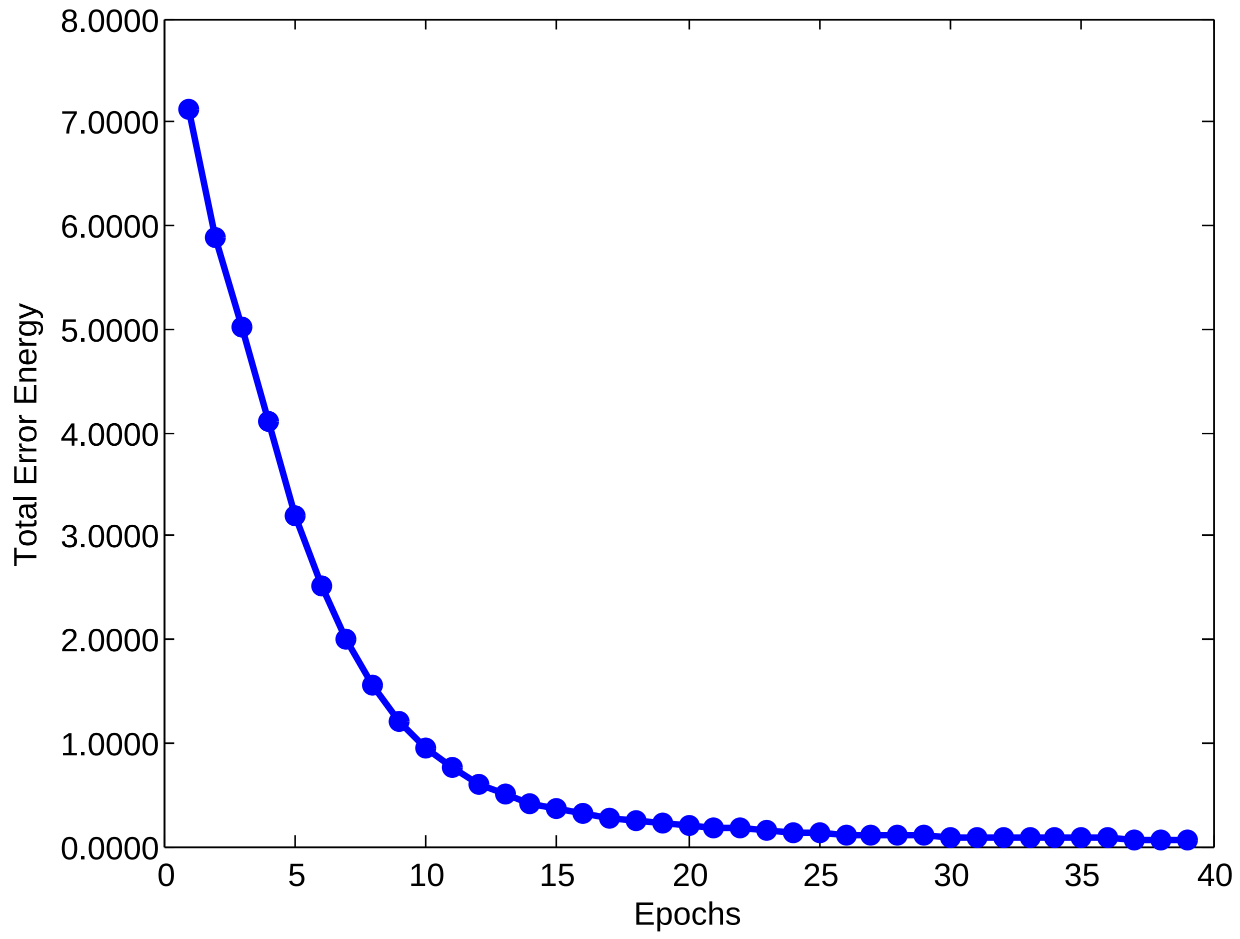


Figure 52: Convergence of Total Error Energy During the Training Process

classification rate is over 96%. It is shown that the implemented ANN system can successfully identify the expected output patterns, given moderate training efforts. Considering the 16-bit data representation format used in our whole design, we may achieve considerably better performance if more hardware resources are available as the semiconductor device dimension keeps shrinking.

The second experiment we conducted is primarily to testify the performance of the implemented ARANN system using a function approximation case. In section 3.3, motivated by the increasingly growing demands on non-invasive neuroprosthetic techniques for improving the functional movements of paralyzed and disabled individuals, we presented a multilayer

ANN-based model to explore the inherent correlation between the intrinsic impaired neuromuscular activities of people with spina bifida (SB) and their extrinsic locomotion behaviors. The goal was to investigate the feasibility and practical implementation issues of applying ANN theories to develop a closed-loop neuroprosthesis or movement assistive systems.

The study was conducted based on twelve individuals with lumbar or sacral level spina bifida (12 trails for each of 12 participants). We averaged the results from these 12 trails and separated the left and right gaits. Thus we obtained totally 24 experimental datasets, each of which contains leg neuromuscular activities and end-point locomotion parameters. The inputs investigated for the proposed ANN model are electromyography (EMG) data and the co-activation indexes for each muscle pair. EMG data collected from the subjects using surface electrodes includes *tibialis anterior* (T), *gastrocnemius* (medial head, G), *soleus* (S), *quadriceps* (rectus femoris, QR; vastus lateralis, QV), and *hamstrings* (biceps femoris, H). Accordingly, the co-activation indexes were calculated for each muscle pair, T and G, T and S, QR and H, QV and H, G and QR, G and QV, S and QR, as well as S and QV. Thus, there are totally 14 inputs including each normalized muscle burst duration and muscle co-activation ratio. On the other side, the output variables were 6 normalized end-point locomotion parameters (stride length, step width, stance phase ratio, double support phase ratio, step cadence (steps per minute), and stride velocity). Due to the differences in leg length among individuals, the gait parameters related to this factor needed to be normalized by leg length [35, 36].

Figure 53 illustrates the converged training error curve. It is shown that the training process can achieve a reasonably small error and converge into a more stable status within 200 epochs. We evaluate the prediction power of ARANN system on all 6 end-point locomotion parameters and demonstrate its training effects in Figure 54, where the red dotted line represents the actual values obtained from the laboratory experimental measurements and the blue line indicates the predicted values given by ARANN system. Figure 54(a) shows the initial status of neural network without training, that is, all synaptic connections are initialized with randomly generated weights. It is manifest that the current neural network system provides meaningless results that are significantly skew away from the actual values. In contrast, Figure 54(b) presents the final status of neural network after training. When all

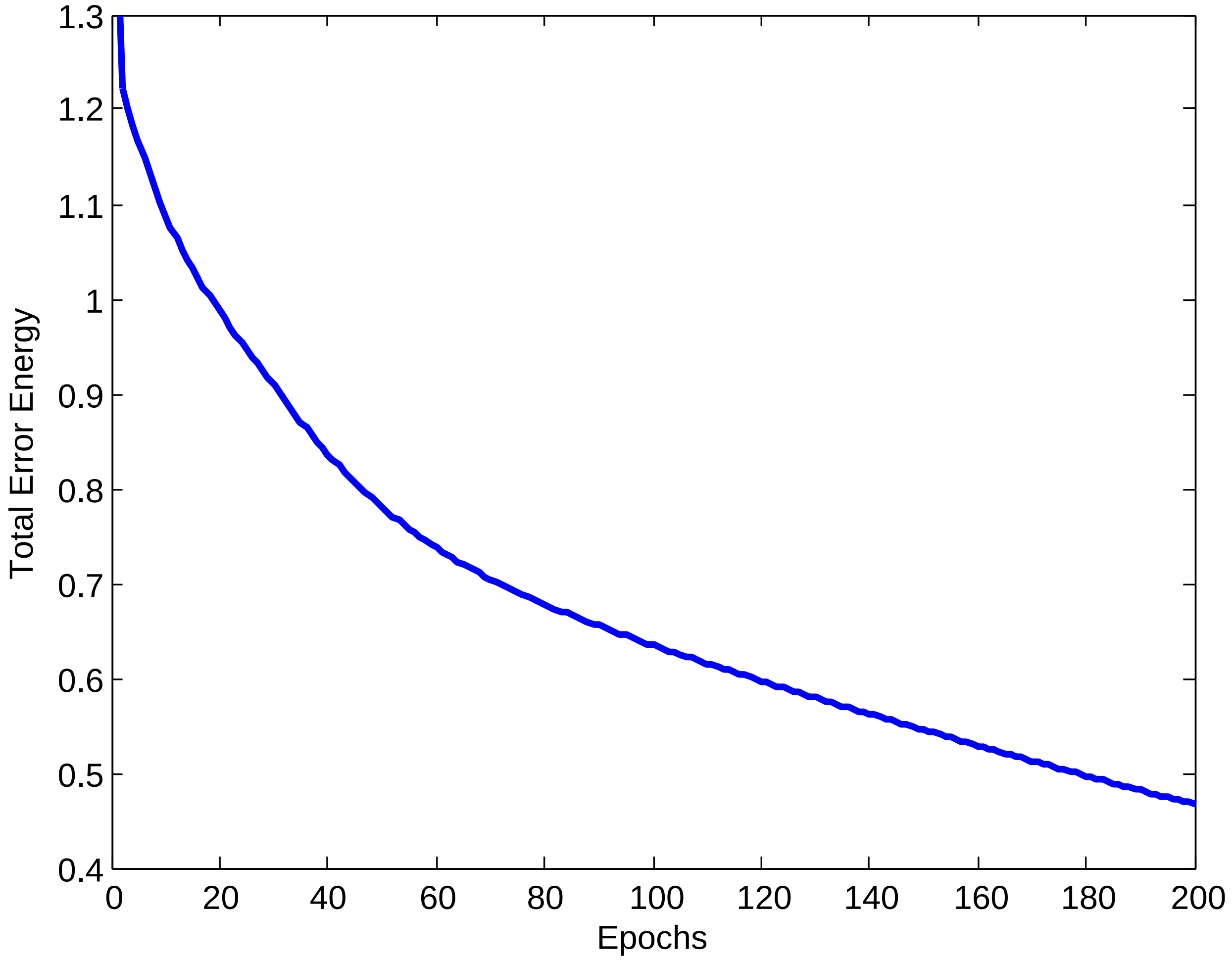
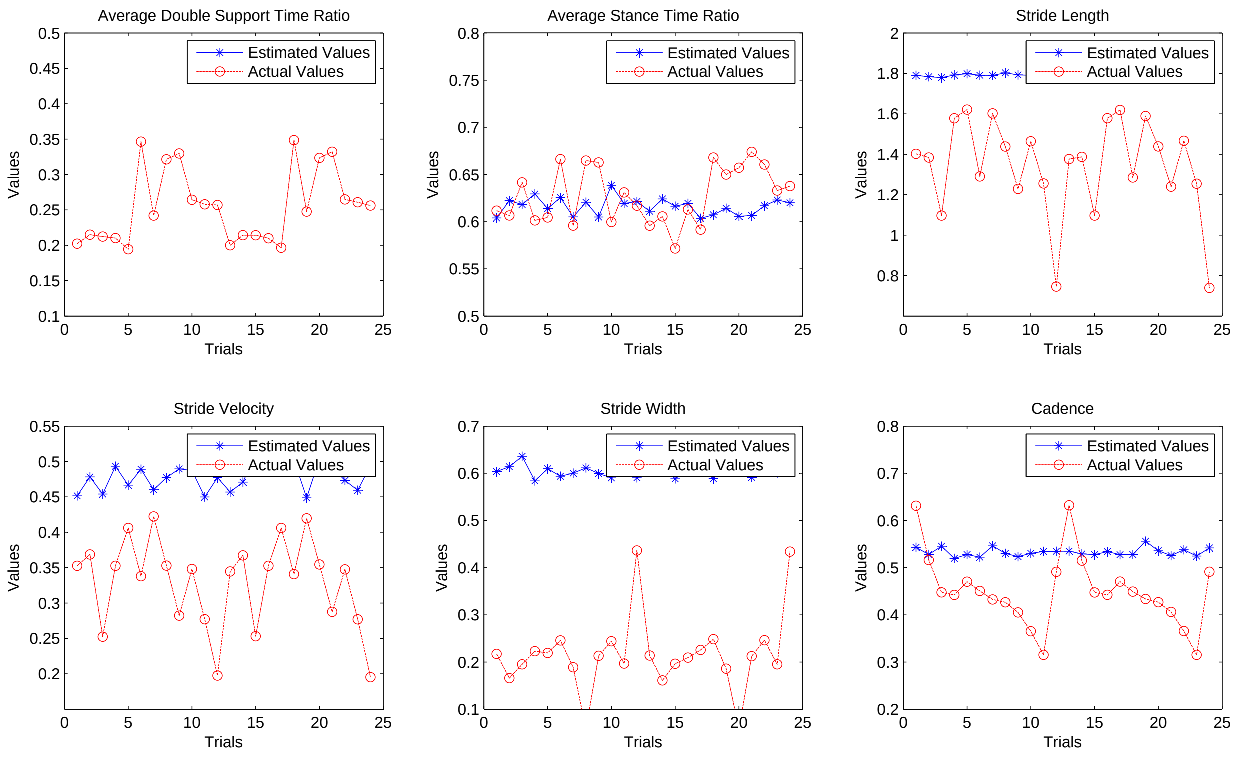
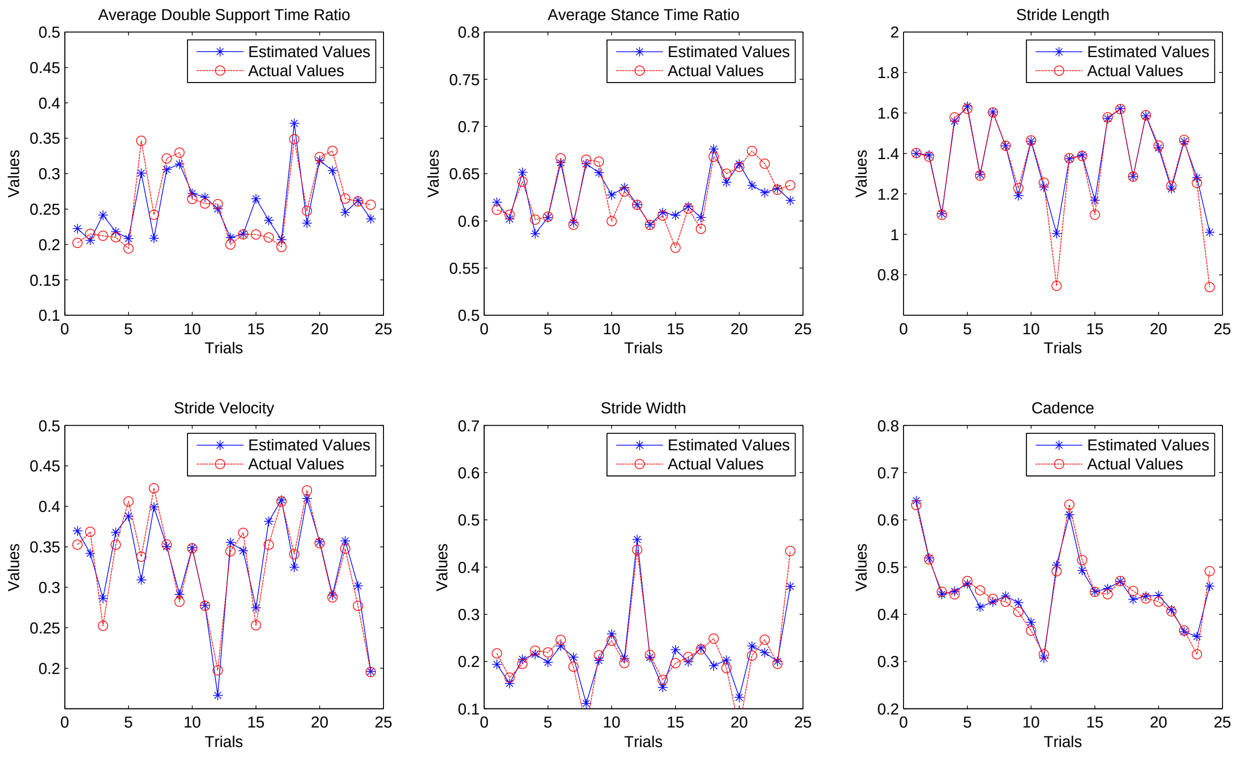


Figure 53: Training Error Convergence of the Endpoint Locomotion Prediction Case

of weights and biases associated with synaptic connections are well tuned and the ARANN system is well established, we find that the predicted end-point locomotion parameters are closely matched with their actual observed values. The fact that the prediction performance is satisfactory across all 12 subjects, further reveals the unbiased and homogeneous features of artificial neural networks for function approximation problems.



(a) Before Training



(b) After Training

Figure 54: Prediction Performance of ARANN for Endpoint Locomotion Prediction Case

6.3 SELF-HEALING PROCESS OF ARANN

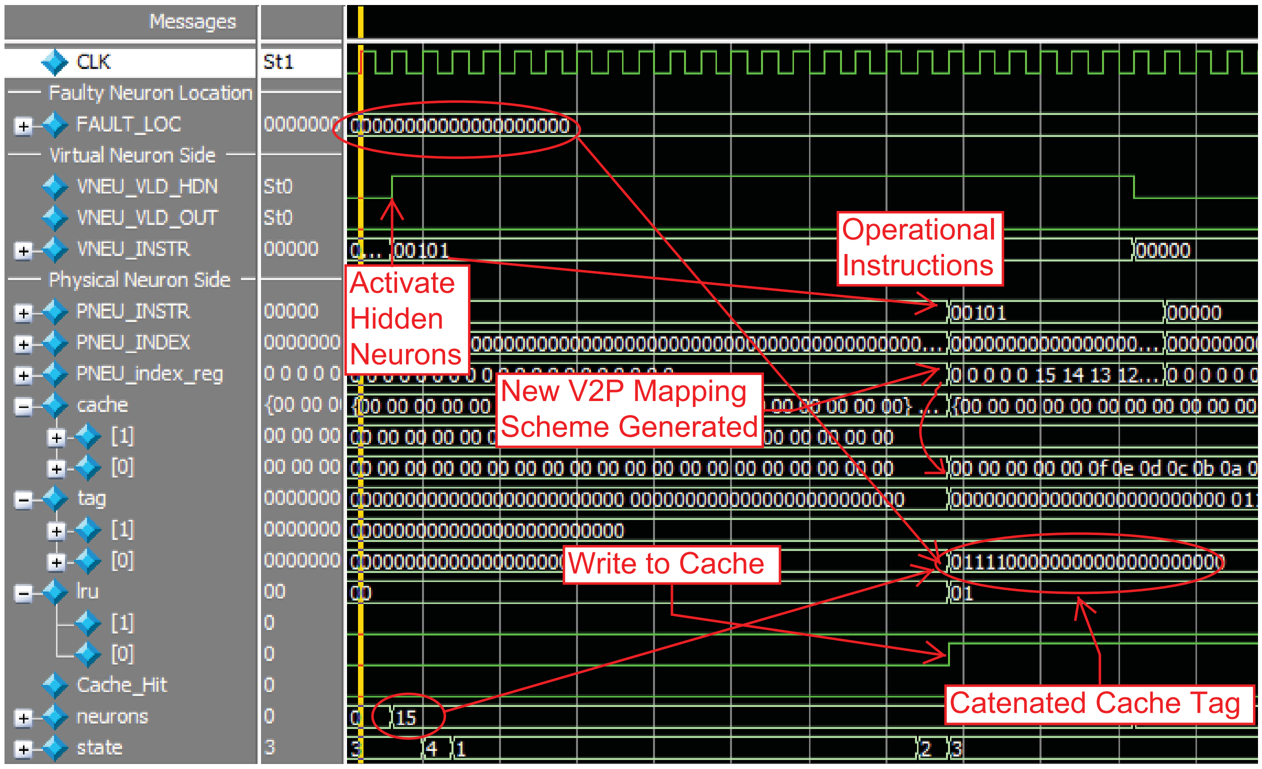
To better utilize the massively parallel processing nature of neural networks and facilitate their structural adaptation, we propose a *Distributed Artificial Neural Network (DANN)* architecture. Also, given the loosely coupled computations supported by DANN, we present a novel *Decoupled Virtual-to-Physical (V2P) Neuron Mapping* strategy to implement a cost-effective system adaptation scheme. Specifically, we propose a "neuron virtualization" by abstracting away the direct connections between ANN controller and all physical neuron units, and inserting a flexible V2P neuron mapping block to determine appropriate connections between virtual and physical neuron ports, according to the desired number of virtual neurons specified by the algorithm and the availability of individual physical neurons implemented on the hardware. Supported by the highly flexible and reconfigurable hardware infrastructure, the ARANN system is capable of adapting ANN's structures and operations, to either meet the algorithmic needs or react to unexpected faults at any neuron. In this section, we would like to demonstrate how ARANN successfully achieve autonomous adaptation for these two scenarios.

In order to meet strict area requirements of future portable applications, we implement ARANN in a very cost-effective manner. Observing that neuron nodes in either hidden layer or output layer have identical arithmetic functionalities except their different positions in the whole system operational flow, we proposed a bidirectional time-multiplexed neuron-reusable neural network design with the highest degree of resource reuse (section 5.4.1). Specifically, we implemented a generic neuron pool including a set of identical neuron units and dynamically involve some of them into the current operational stage according to the training algorithm. Therefore, the first need to dynamically adapt ANN's structure comes from the algorithmic requirements of ANN's training or functioning processes. When there are any changes associated with virtual neuron ports, such as the stage transitions depicted in Figure 18, the virtual-to-physical mapping module will be triggered to establish a new mapping connection scheme between virtual neurons and physical neurons.

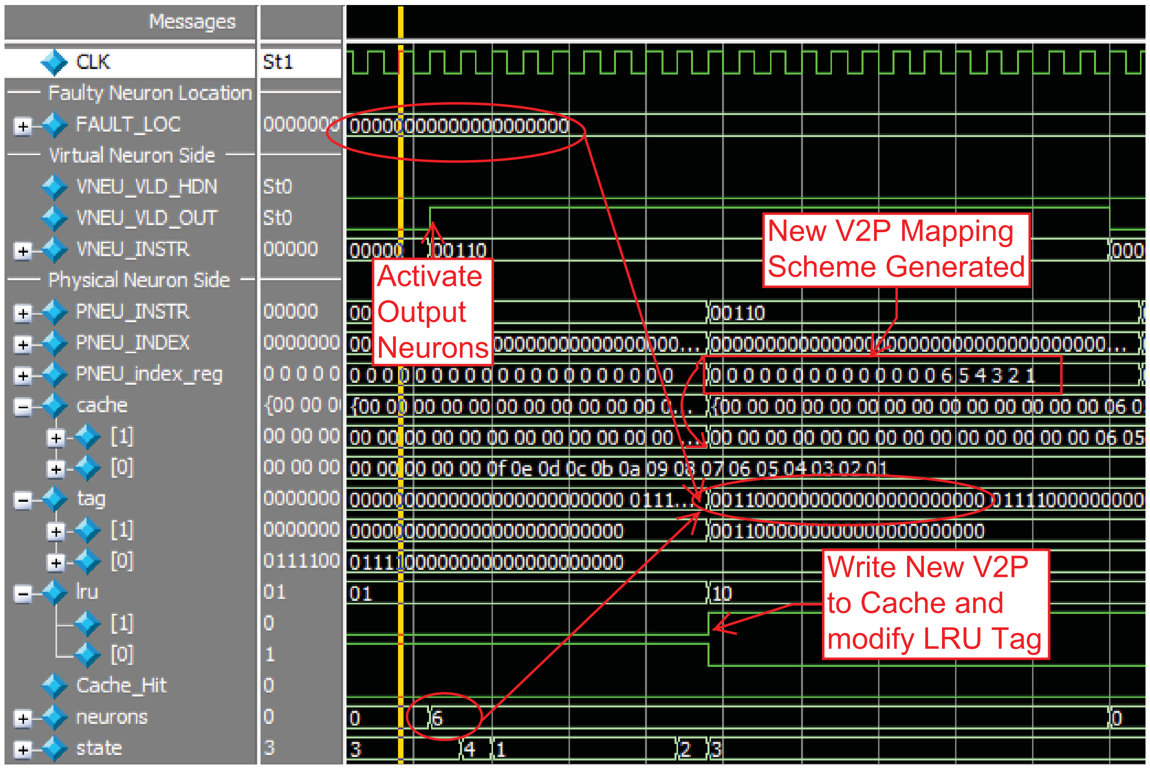
However, such regular stage transitions between hidden layer and output layer (e.g., at least 4 times for one training pattern within an epoch) demand frequent V2P neuron remapping processes and thus will introduce considerable time overhead. Inspired by the hierarchical memory system in state-of-the-art computer architecture, we propose to incorporate a V2P mapping cache to temporarily store the most recently generated (used) V2P mapping schemes. Considering the unique temporal locality of V2P mapping schemes in ARANN, i.e., the system will repeatedly be reconfigured between these two structures (i.e., hidden layer and output layer), the structural adaptations can be simplified into a series of cache access if there is no further needs on a new ANN topology or no new faulty neurons reported.

As shown in Figure 55(a), when ARANN is initiated and firstly goes into the feed-forward computations in the hidden layer (specified by the signal "VNEU_VLD_HDN"), a new V2P neuron mapping scheme is needed for assigning appropriate number of physical neuron units to act as hidden neurons. Once the V2P mapping scheme is determined, the activation signals and the operational instructions from ANN controller will be distributed to those involved physical neurons simultaneously. Accordingly, this new V2P mapping scheme will be also sent to the V2P mapping cache (i.e., cache[0]). Since the determination of current V2P mapping scheme relies on the desired number of neurons as well as the availability of all physical neurons at this moment, these two parameters will be combined together to form a unique tag (i.e., tag[0]) associated with the content item (i.e., V2P mapping scheme) just stored in the cache. Besides that, the Least Recently Used (LRU) bit is marked (i.e., LRU[0]=1) to indicate that this item of cache content is newly established. Similarly, when ARANN switches to the operations in the output layer, as shown in Figure 55(b), another new V2P mapping scheme is needed since the output layer requires different number of physical neuron units. The newly generated V2P mapping scheme and corresponding operational instruction will be distributed to all involved physical neuron nodes. Also, this new scheme will be sent to the cache. Since the Cache Line 0 in the cache has been already occupied by the V2P mapping scheme previously generated for hidden layer, indicated by its LRU bit (i.e., LRU[0]=1), this new scheme will be stored into another line of the cache (i.e., cache[1]) and the corresponding LRU bit is marked (i.e., LRU[1]=1 & LRU[0]=0).

In Figure 56(a), ARANN system switches back to the hidden layer. Since the current V2P mapping conditions exactly match the tag of the Cache Line 0 (i.e., tag[0]), the content



(a) For Hidden Layer

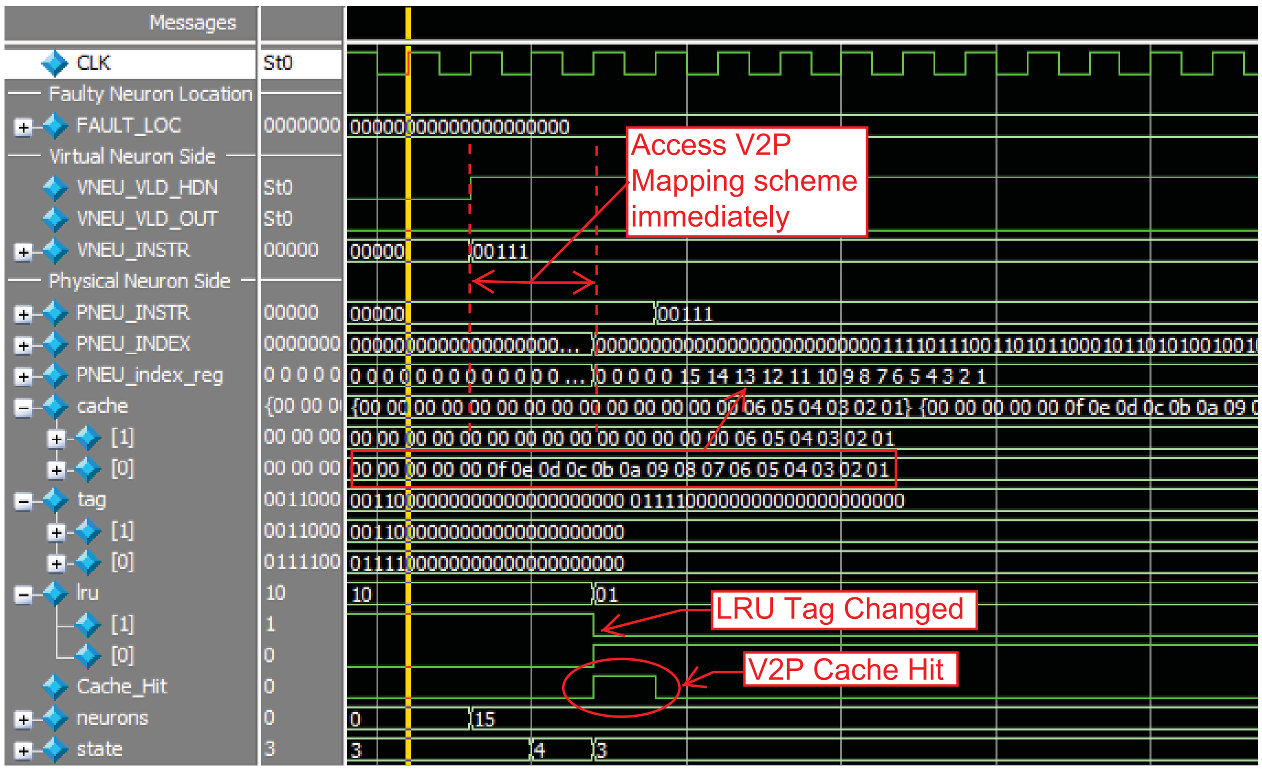


(b) For Output Layer

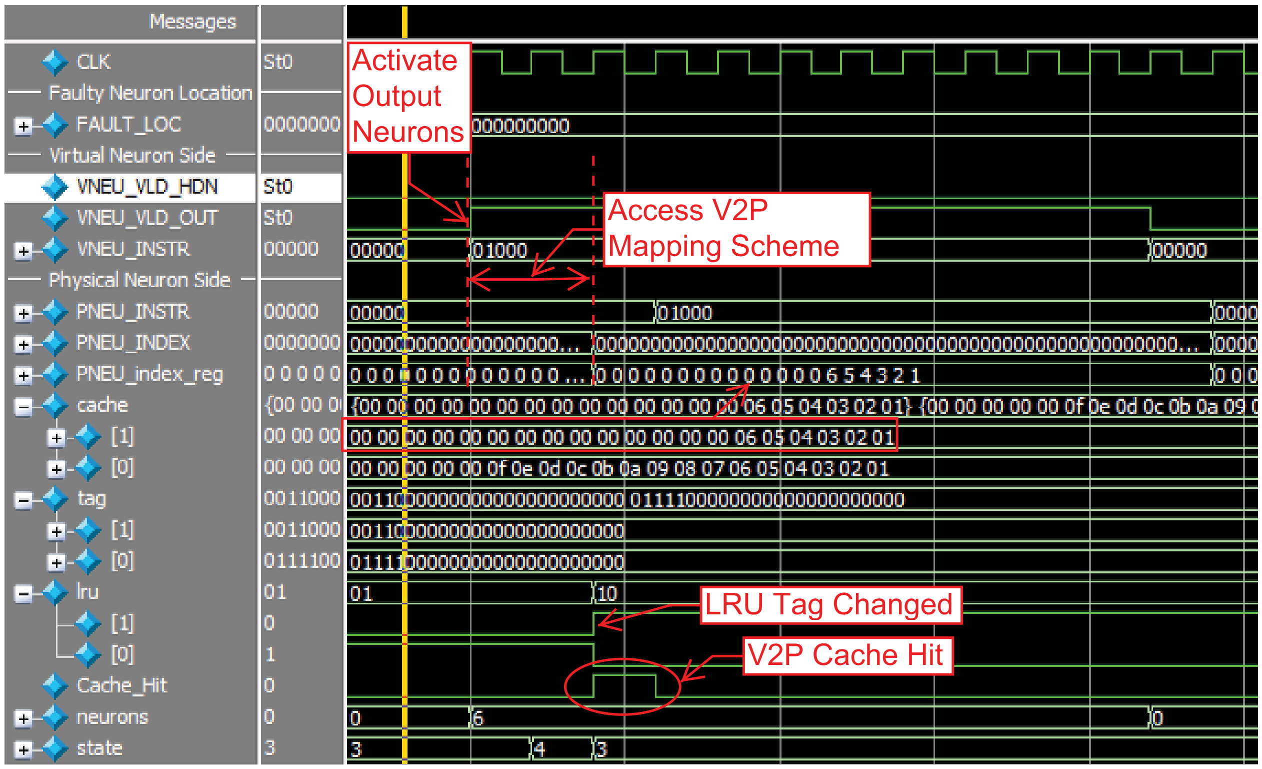
Figure 55: Adaptations of V2P Mapping Scheme for Hidden & Output Layers (Cache Miss)

of Cache Line 0 can be immediately accessed and the V2P remapping effort can be achieved within a significantly reduced time period. Accordingly, a cache hit signal is marked (i.e., Cache_Hit=1) and the LRU bit is set to high again (i.e., LRU[0]=1) because of the recent cache hit. It is shown that such cache-enabled V2P mapper only needs two extra cycles for the structural adaptation in this case, comparing with the dozens of cycles needed for a standard V2P mapping task. Similarly, when ARANN moves to the output layer again, a cache hit is indicated (i.e., Cache_Hit=1) for the Cache Line 1. The new V2P mapping scheme can be also obtained immediately by accessing Cache Line 1, which maintains correct mapping solution generated previously for the output layer. It is shown that such cache-enabled V2P mapper only needs two extra cycles for the structural adaptation in this case, comparing with the dozens of cycles needed for a standard V2P mapping task. The tremendous amount of computation overhead caused by frequently repeated ANN structural adaptations can be significantly alleviated in such a cost-effective manner.

In scenarios discussed above, we have successfully addressed the issue of frequently repeated system structural adaptations. However, in those cases, we assume that all physical neurons are available, which means, there is no fault reported and the bit indicating the availability of each neuron should be zero (i.e., FAULT_LOC=0b0000_0000_0000_0000_0000 for 20 physical neurons). In what follows, we will investigate the ARANN system behaviors in case one or more faulty neurons are detected. In Figure 57, the ANN system is currently running in the hidden layer and the corresponding system structure is determined by the V2P mapping scheme which is accessed from the cache. At a certain moment, several faulty neurons are identified and their specific locations are reflected in the FAULT_LOC signal (i.e., marked as "1" in FAULT_LOC). The change of the FAULT_LOC signal immediately triggers the remapping process in the V2P Mapper. After a number of cycles, a new V2P mapping scheme is determined and all those faulty neurons indicated by the FAULT_LOC signal have been successfully disconnected and isolated. Three spare ("unused") physical neuron units now are activated and swapped into the system operations. Since the current V2P mapping scheme is generated based on a scenario different from what we met before, it will be stored into the V2P mapping cache to replace one of cache lines according to their LRU bits.



(a) For Hidden Layer



(b) For Output Layer

Figure 56: Adaptations of V2P Mapping Scheme for Hidden & Output Layers (Cache Hit)

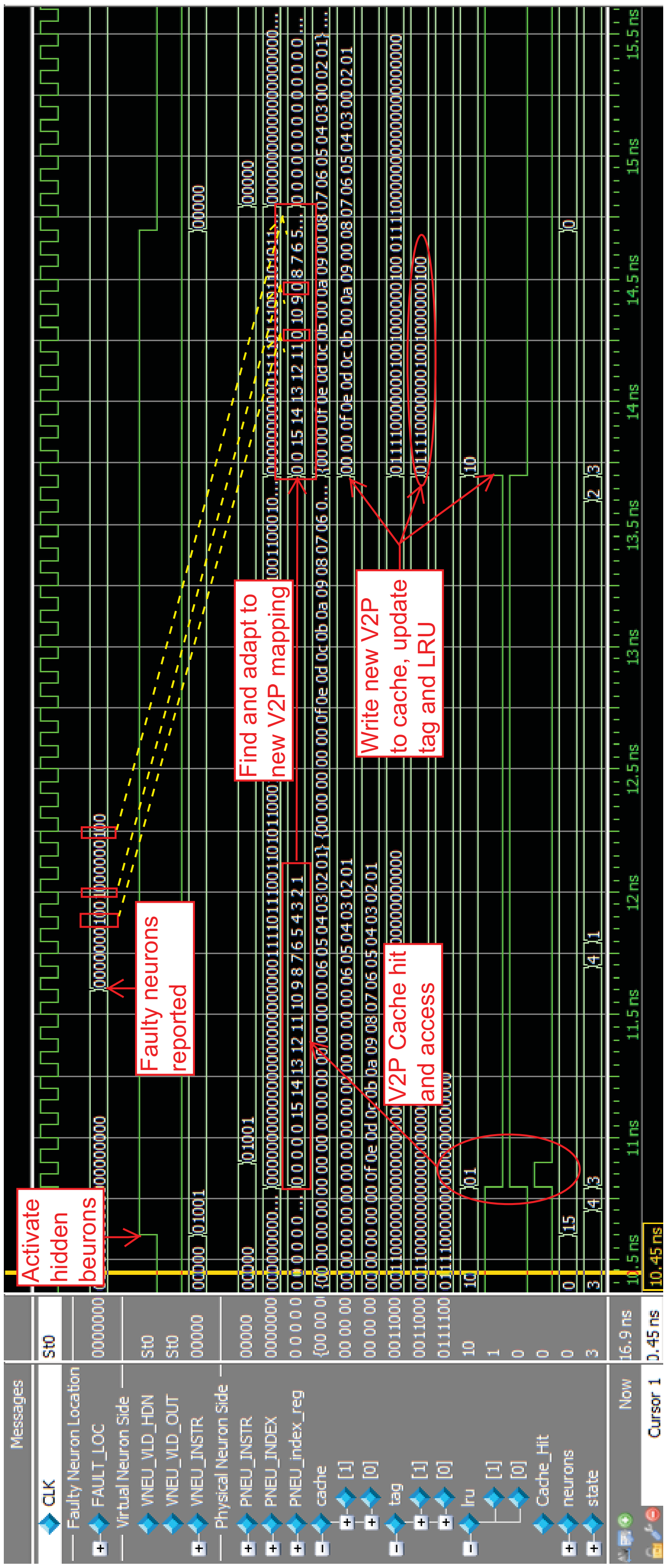


Figure 57: Adaptation of V2P Mapping Scheme in case of Faulty Neurons Detected

Given the scenario presented in Figure 57 where the unexpected faults occurring on one or more neurons automatically trigger the system structural adaptation to maintain appropriate operational level, we would like to investigate the whole training process of the proposed ARANN, which involves the system structural adaptations both requested by the repeated stage transitions between hidden layer and output layer, and caused by newly detected faulty neuron units. As shown in Figure 58, in this case, the faulty neuron string FAULT_LOC indicates two damaged neuron units: Neuron 1 and Neuron 14, while the system requires 18 hidden neurons and 12 output neurons. According to the desired number of neurons and the availability of each physical neuron, the V2P Mapper establishes appropriate connections for virtual neuron ports managed by the ANN Controller and physical neuron units. The Neuron 2 to 13 will be constantly activated because they will be used for computations in both hidden layer and output layer. In contrast, the Neuron 15 to 20 will be periodically enabled when the neural network needs to perform the operations in the hidden layer. The Neuron 1 and Neuron 14 are constantly disabled due to their faulty statuses. The decreased training errors associated with each train epoch demonstrate that the ARANN system can maintain appropriate operational level and still train itself toward a more stable and capable status, even though some neuron nodes have been damaged by unexpected events.

From Figure 55 to 58, we have demonstrated ARANN's remarkable capability of automatically adapting its structures and behaviors to meet the requirements of resource-efficient neuron reusing and react to unanticipated neuron failure. For neural network systems, another major concern is whether the network can still perform correct operations and maintain appropriate behaviors, in case the system is suddenly interrupted by a newly detected neuron failure and thus its topological structure has to be reconfigured to isolate the faulty neurons. As we elaborated in section 4.3.3, this is particularly critical for the back-propagation training process, which usually involve a tremendous amount of data write/read access and the closely intertemporal data dependency (i.e., the data dependence between current stage and subsequent stage in either feed-forward or back-propagation process). Given the proposed *Dual-Layer Memory Synchronization* mechanism, we would like to achieve a smooth, accurate and consistent recovery of neural network systems no matter when an unexpected fault is detected. Thus, in what follows, we will investigate if the proposed ARANN is truly

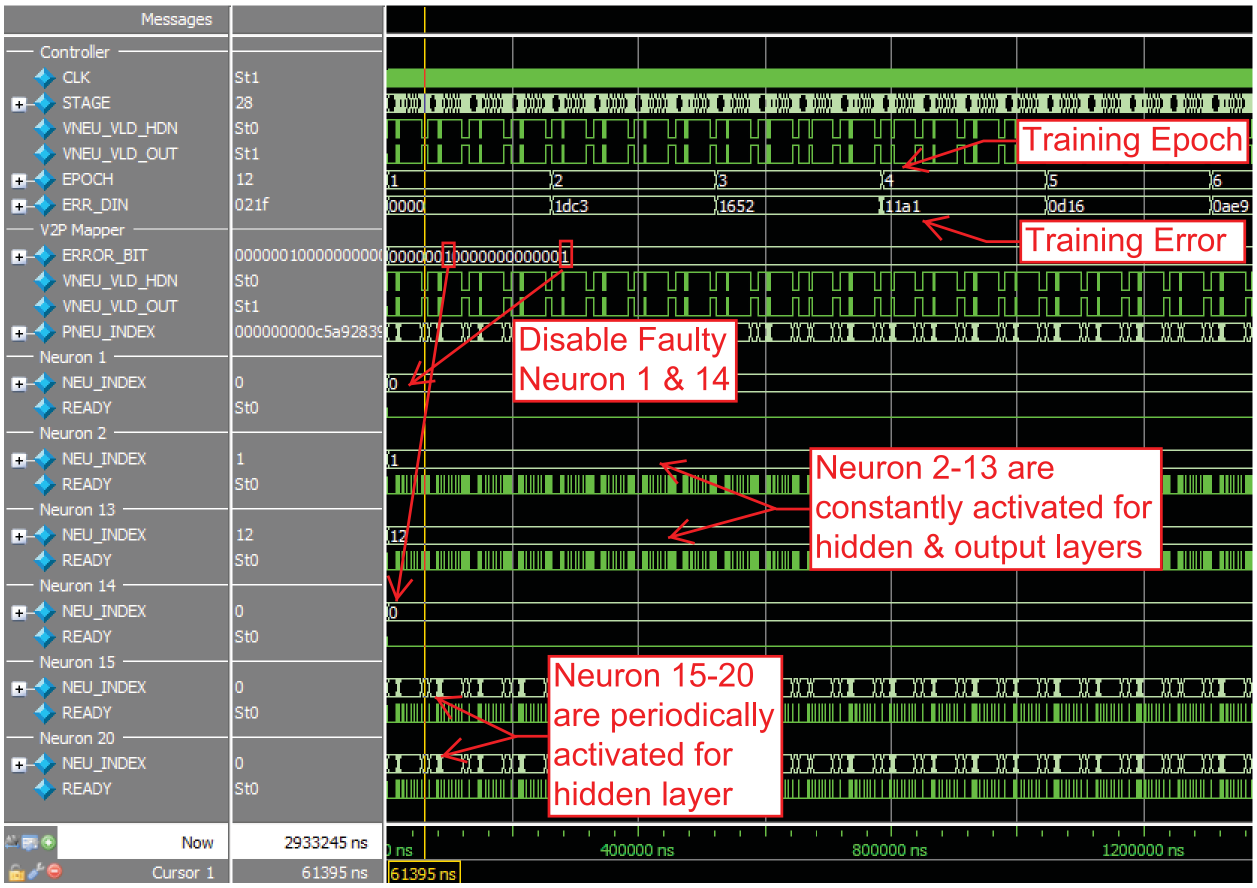
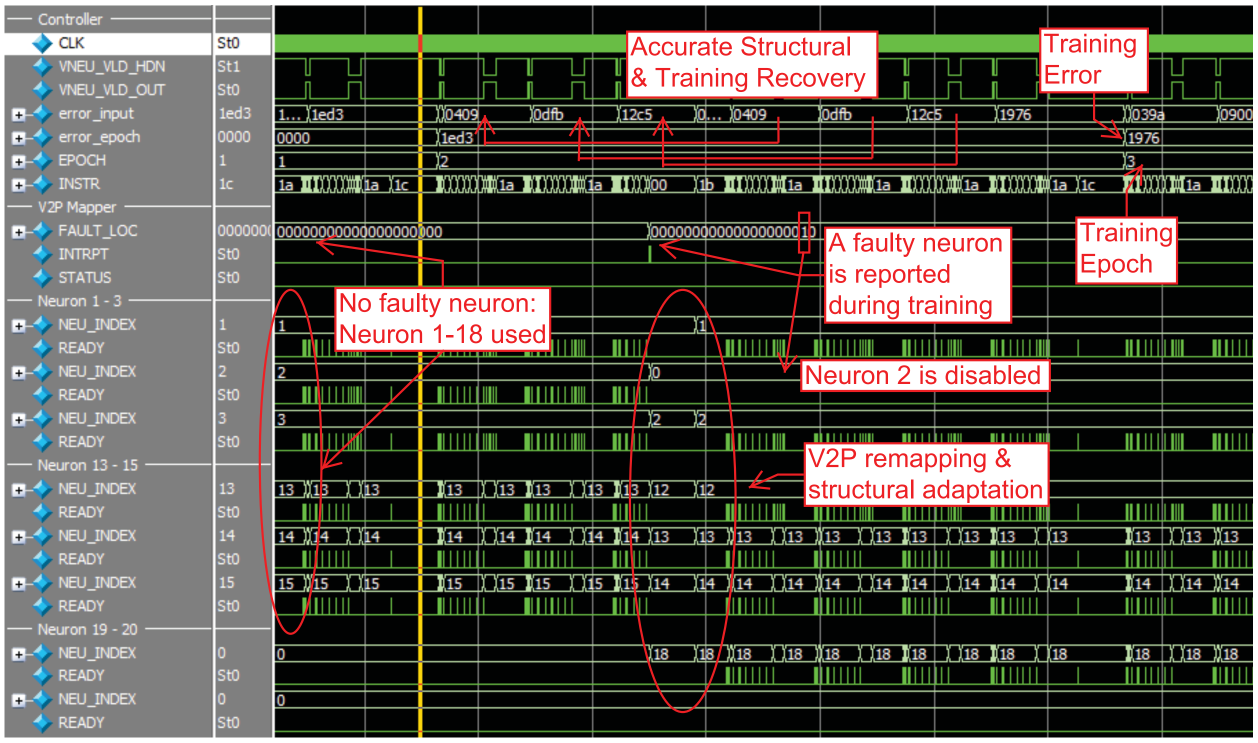


Figure 58: Training Process and Neuron Allocation of ARANN with Faulty Neurons

capable of autonomously adapting its own structure while keeping the system operations on the right track in response to unexpected events.

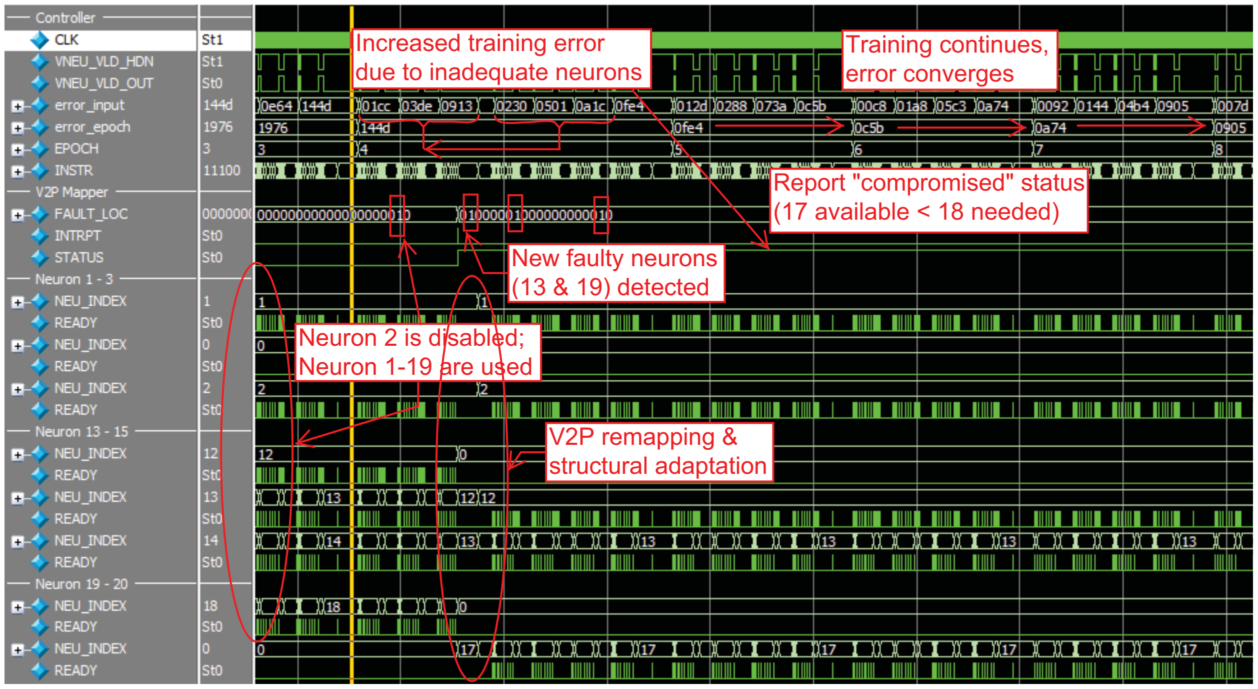
In Figure 59(a), as indicated by the left circle, the ARANN initially uses 18 neurons according to the specified neural network algorithm and topology (i.e., 18 neurons in the hidden layer and 12 neurons in the output layer). Assuming there is no faulty neuron reported at this moment (i.e., the variable FAULT_LOC is filled with all zeros), the Neuron 1 to 18 are activated and involved into the current neural network. The neural network is thus able to perform appropriate operations for the training purpose. Unfortunately, one faulty neuron (e.g., Neuron 2 in this case) is detected and reported at a certain moment during the training. Accordingly, an interrupt signal (INTRPT) is immediately triggered and the reconfiguration of the virtual-to-physical neuron connections is desired. Once the V2P mapper works out a new V2P mapping scheme, the corresponding neuron units are activated and involved into the operations of neural network. For example, Neuron 19 is enabled to compensate for losing the Neuron 2, which has been successfully disabled and isolated from the main network. More importantly, since the faulty neuron is detected and reported right in the middle of training process, the faulty neuron inevitably has already introduced "contaminated" data into the system, which may also has been accessed by other neurons. The system will retrieve the correct training information and recover its training process by synchronizing network's synaptic configurations with the Level-2 Synchronization Memory, which keeps the latest correct network configuration from previous training epoch. It is shown that, even after system structural adaptation, the ARANN can still return back to the right training track (exactly same training error for each input pattern) with new group of neuron units. The underlying reason of such accurate and consistent recovery is that the system has successfully passed all of its latest correct configuration information (including t he information generated previously by the faulty neuron) to all currently involved neuron units. This feature makes ARANN a superior solution providing both the infrastructure recovery and the behavioral restoration.

Now the neural network is running with one faulty neuron unit (Neuron 2). Unfortunately, at another moment, two faulty neurons are detected and reported again (e.g., Neuron 13 and 19) as shown in Figure 59(b). Accordingly, ARANN has to adapt its structure again



Accurate Structural & Training Recovery
Training Error
A faulty neuron is reported during training
Training Epoch
No faulty neuron: Neuron 1-18 used
Neuron 2 is disabled
V2P remapping & structural adaptation

(a) Adequate Spare Neurons (Accurate Training Recovery)



Increased training error due to inadequate neurons
Training continues, error converges
Report "compromised" status (17 available < 18 needed)
New faulty neurons (13 & 19) detected
Neuron 2 is disabled; Neuron 1-19 are used
V2P remapping & structural adaptation

(b) Inadequate Spare Neurons (Compromised Training Recovery)

Figure 59: Recovery of Training Process In Case of Fault-Triggered Structural Adaptation

to meet the needs of neural network algorithms. However, the system cannot find a completely satisfied V2P mapping scheme this time, because there are only 17 healthy neurons available on the hardware, less than the 18 neurons required by the algorithm. In this case, the V2P mapper continues to work out an "optimal" V2P connection to incorporate as many neuron units as possible into the network. Beside that, the V2P mapper will also trigger an operational status signal (STATUS) indicating that the system now is running in a "compromised" manner with less computational nodes. Given the fact that there are not adequate neurons involved in the neural network, it is no longer possible to restore the training process to the exactly same track as the previous system configuration without faulty neurons. For instance, the training errors in the current epoch have been increased from $(01cc)_{16}$, $(03de)_{16}$, and $(0913)_{16}$ to $(0230)_{16}$, $(0501)_{16}$, and $(0a1c)_{16}$ respectively. Such increases also reflect the change of network structure, that is, under the same synaptic configurations, a neural network with less neurons would likely generate larger error than the one with more neurons. Although the training process cannot be restored to the previous track due to the loss of neuron units, the "compromised" neural network can still train itself to minimize the training error, as indicated by the gradually converged errors of each training epoch.

In what discussed above, we have particularly focused on the training process of ANN and explicitly illustrated how the proposed ARANN reacts to the unexpected neuron failures, achieves the functional recovery, and thus maintains the appropriate operational level through cost-effective dynamic system structural adaptations. The main reason that we have to carefully manipulate ANN structural adaptations during training is due to the fact that ANN training process is a highly intricate and complicated procedure, involving a high degree of intertemporal data dependency between each computational stage and a tremendous amount of data communications among all neuron units. Comparing with highly involved back-propagation training process, the functioning of ANN, either used for classification or function approximation, is much simpler because it only involves the feed-forward calculations and does not change any parameters associated with the ANN synaptic configuration. Thus, ANN's functioning process can be regarded as a direct subset of the training procedure and the corresponding system recovery mechanism reacting to neuron failures could be much more straightforward.

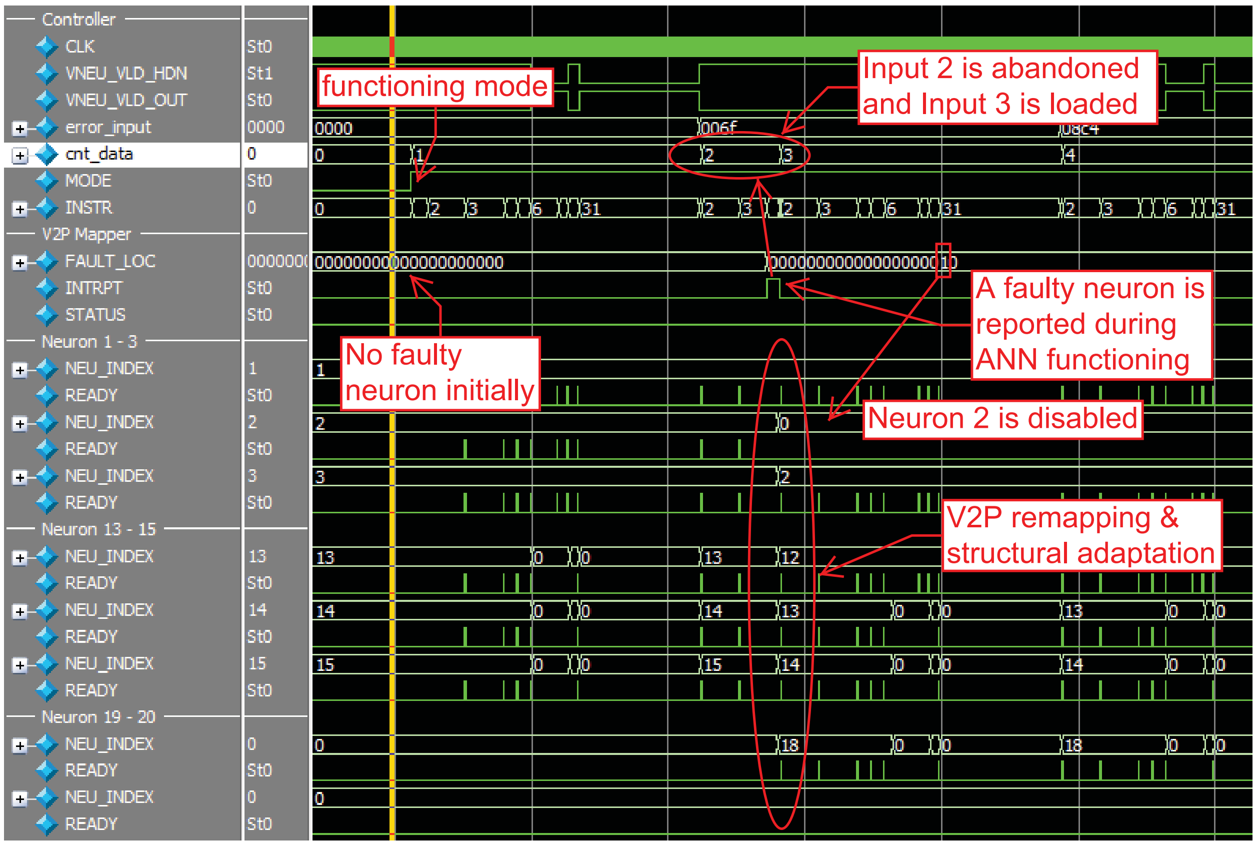


Figure 60: ANN Functioning Process and Neuron Allocation of ARANN with Faulty Neurons

Fortunately, the highly involved training process of ANN is only needed for certain cases, where ANN needs to be re-trained to address different problems or to incorporate new training data set. The majority of system operations are primarily made up of those relatively simple feed-forward functioning processes. Accordingly, it is significant to further investigate how ARANN reacts to the neuron failures which occur during the ANN functioning. As shown in Figure 60, we still assume that initially all physical neuron units perform well and there is no faulty neuron detected within the ANN system. The ANN now is running in the "functioning" mode, as indicated by the high "MODE" signal. When the system is processing the Input 2, the failure of a neuron unit is detected and reported to the ANN controller (i.e., Neuron 2 in this example). The newly detected faulty neuron immediately triggers the structural adaptation of the ANN system, as what we elaborated previously. The result of such structural adaptation is to disconnect the faulty neuron (i.e., Neuron 2) from the neural network and integrate another available neuron unit (i.e., Neuron 19) into the network to meet the system requirements regarding the desired number of neuron nodes. Since the faulty neuron is detected when processing the Input 2, the current calculation for the Input 2 has been contaminated and disrupted due the occurrence of neuron failure. In this case, one convenient way is to discard the current input pattern and load a new one into the ANN system, because the use of ANN in biomedical applications, particularly physiological signal analysis, is essentially an stochastic process. That is, the determination of any certain medical condition or symptom is based on a set of (or a series of) ANN functioning results, rather than the generated result for any single input pattern. In Figure 60, we observe that the ARANN can timely remove the faulty neuron from the main network and successfully recover the system operation by swapping in a new neuron unit within a very short period. After the system structural adaptation, the following input patterns can be fed into the neural network for continued processing.

Finally, given all scenarios discussed above, it is manifest that the proposed ARANN architectural framework shows superior capabilities in both maintaining the system operations by autonomously adapting network's structure and configuring the connections of all physical neuron units, and achieving fault-tolerant neural networks by exploring a optimal trade-off between the functioning performance and resource availability.

6.4 SELF-OPTIMIZING PROCESS OF ARANN

Along with the remarkable efforts researchers have made to discover more effective ANN algorithms for some as of yet unsolved problems, another important research question of great concern is how to find and determine the best structure and configuration for a given ANN algorithm. Actually, this is a far more efficient way to utilize ANN's incomparable computational capabilities. To solve real-world problems using ANNs, it usually requires the use of highly structured networks of a rather large size. A rule of thumb for obtaining good generalization capability is to use the smallest system that will fit the data. Not to mention that using oversized neural network that carries too many redundant or less influential computational nodes (neurons) will significantly increase the burdens on power consumptions, which is prohibitive to meet the strict requirements of emerging ANN-based portable systems. It is well agreed that deploying ANNs onto hardware platforms is a rather challenging task, due to the tremendous amount of intricate data computations and communications within neural network as well as the large number of neuron nodes usually involved. Another major motivation to develop a flexible neural network platform with the capability of adapting and optimizing its structure in an autonomous manner is the increasing demands on the more diversified neural network systems, which means to help ANN choose an appropriate structural configuration according to specific performance constraints and design trade-offs between functioning accuracy and complexity overhead.

Leveraging the reconfigurable and adaptable architectural infrastructure provided by ARANN, we incorporated the concept of neural network pruning into ARANN and proposed a *Self-Optimizing Artificial Neural Network (SOANN)*, making use of ARANN's incomparable capabilities of connecting and disconnecting any physical neuron unit to/from the main network on the fly (refer to section 4.4). Instead of determining an "optimal" neural network structure for one certain application by the off-line analysis, the ARANN architecture will be able to evaluate the hybrid system cost involving both functioning accuracy and complexity overhead, and then adaptively explore the most optimal network structure with the appropriate performance tradeoff. Specifically, the system will start from the default network structure (with all available neurons or an estimated number of neurons believed to be large enough for the current application) and train the current network, following the standard back-propagation training procedures. Once the current network has been well trained, its cost will be evaluated according to the hybrid system cost function. Then the ARANN will automatically prune one neuron node from the main network and repeat the previous training and cost evaluation processes. This procedure will continue until an “optimal” neural network structure with the minimal system cost is found.

Figure 61 and 62 illustrate the heuristically self-optimizing processes of ARANN system from the network structure with 20 neurons in the hidden layer to the network with only 1 neuron. The only difference between these two cases is their different training termination criterion: the former uses a fixed number of epochs as the termination criterion for the training of each neural network structure; while the latter sets a more objective and cost-effective termination criterion for all network structures, where the training will be terminated only if the change rate of training error has achieved a reasonably small value (in this study 1% threshold is used). It is shown that the network with less neurons usually has a faster training convergence trend, comparing with the network with more neurons. The underlying reason for this phenomenon is simple network structures are like to involve much less synaptic parameters that need to be adjusted and tuned during training. In contrast, a more complicated neural network with a large set of neurons usually involves extremely intricate parameter adjustments and co-optimizations to best fit the *a priori* training patterns. It is worth mentioning that the standard self-optimizing process starts from the default network structure and ends at a structure with the smallest system cost, rather than the extreme case of searching from the maximum amount of neurons to only one involved neuron, shown in these two figures.

Figure 63 presents the performance costs of a group of neural networks containing various number of neurons, based on different system cost functions. The first case (blue line) is that only the functioning accuracy of neural networks is considered in the system cost function, which is also the most common criterion used by most of ANN users to evaluate their ANN systems. Not surprisingly, the results show a gradually increased system cost (i.e., functioning error) as the number of neurons involved into the neural network decreases. Accordingly, the best choice here seems to be involving all neurons available in the system to

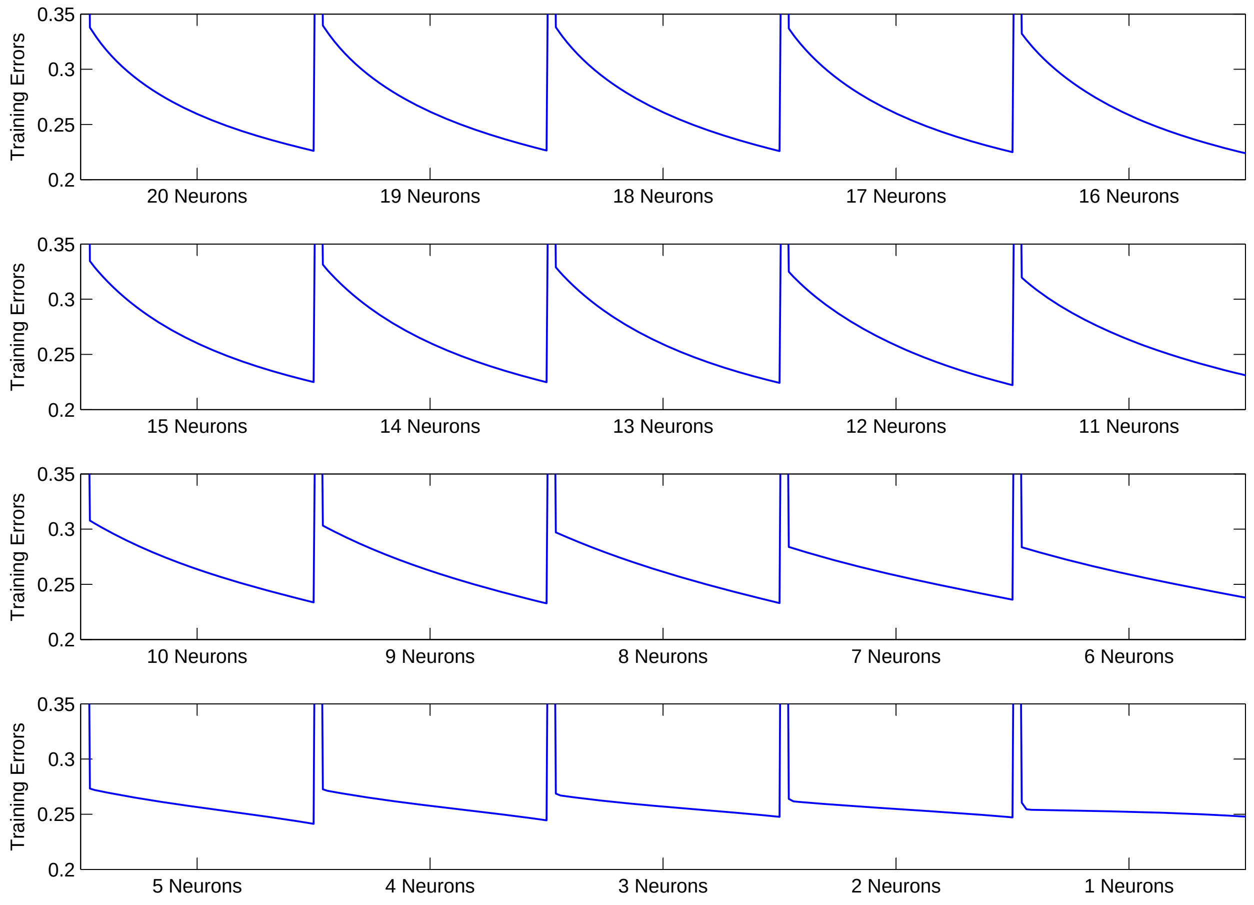


Figure 61: Self-Optimizing Training Process of ARANN with from 20 Neurons to 1 Neuron in the Hidden Layer (Note: Each training contains 50 epochs.)

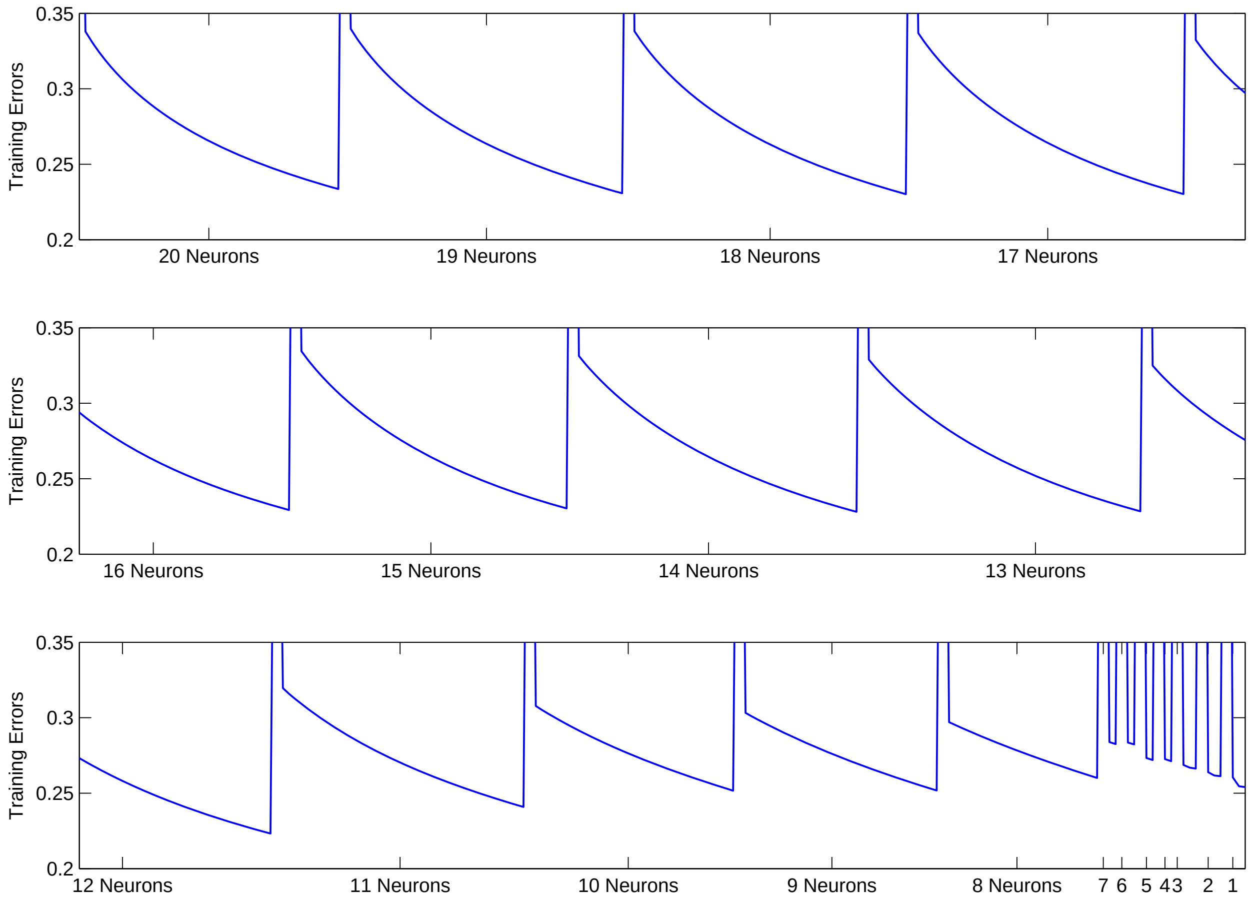


Figure 62: Self-Optimizing Training Process of ARANN with from 20 Neurons to 1 Neuron in the Hidden Layer (Note: Each training is terminated by a reasonably small change rate of training error.)

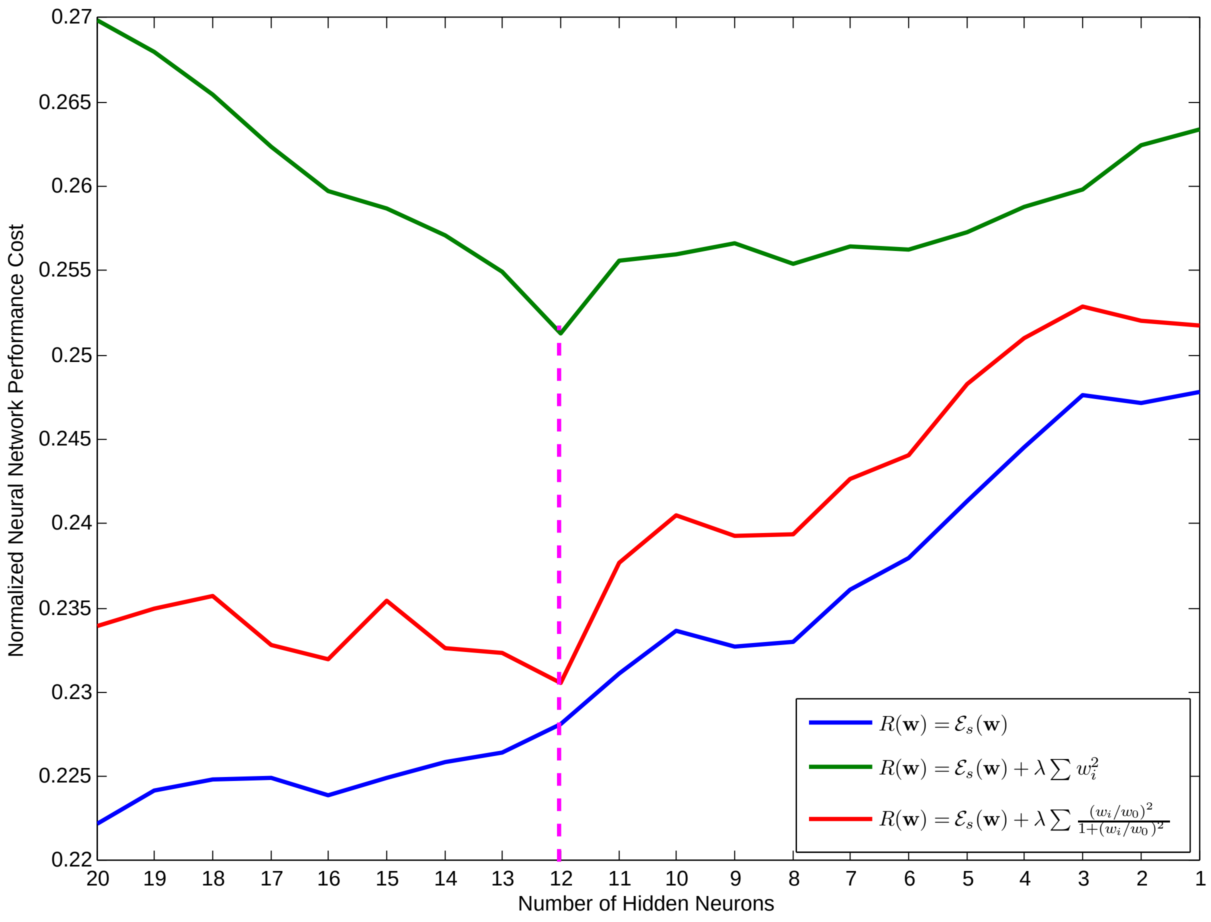


Figure 63: Hybrid Performance and Overhead Evaluation in ARANN Self-Optimizations from 20 Neurons to 1 Neuron in the Hidden Layer

achieve the least performance cost. This trend is coincident with the hypothesis that a neural network containing more neuron nodes usually has better flexibility and more capability to "learn" more complicated knowledge or patterns. However, this will be not the case when a rather large number of neurons have been involved into a neural network, which may cause degraded functioning accuracy.

The other two cases all incorporate certain type of complexity penalty term into their system cost function, besides the functioning accuracy term discussed above. The red line shows the hybrid system costs associated with each neural network structure, based on the so-called weight decay complexity penalty term defined in Equation eq:WeightDecay, where $\mathcal{E}_{\text{total}}$ refers to all the synaptic weights in the network. The integration of weight decay complexity penalty term into overall cost function will help the MLP network trim some synaptic connections that have little or negligible influence on the network and thus remove the neuron nodes that have limited contributions to maintain the functioning capability of neural networks. It is shown that the neural network can achieve a smallest performance cost when there are 12 neurons used in the hidden layer.

Similarly, the green line represents the system costs based another complexity penalty term defined in Equation 4.3, where w_0 is a preassigned parameter. Although the green line shows a slightly different cost trend against the red line, it also presents the smallest system cost when the neural network containing 12 neurons in this case. Comparing with the "optimal" structure given by the system cost function that is only concerned about the functioning accuracy of neural networks, those two hybrid cost functions all select a much simpler neural network structure which could save 8 neurons. This figure demonstrates the feasibility and effectiveness of taking the structural complexity into consideration when evaluating the overall performance/cost of a neural network, to autonomously explore an "optimal" network structure and achieve a "balanced" tradeoff between functioning accuracy and complexity overhead on the fly.

Unfortunately, there are still many research problems associated with such self-optimizing methodology. For example, the selection of an appropriate complexity term always is always arguable. Although the two complexity terms illustrated in this study are among the most effective and efficient metrics used in neural network society, given their relatively simple calculations and the applicability particularly for hardware implementation, there has not been any well recognized criteria yet to objectively and precisely evaluate the "cost" introduced by the redundant complexity. Another major concern about this self-optimizing approach is the *ad hoc* selection of those parameters in the system cost function, such as the *regularization parameter* λ representing the relative importance of the complexity penalty term with regard to the functioning accuracy term. Considering two extreme cases: 1) when the λ is zero, ANN's training process is completely driven by the pursuit of maximum performance (minimum error energy); 2) when the λ is infinitely large, in contrast, now the training of neural networks will be stringently constrained and determined by the desired system complexity. Therefore, the process of exploring an "optimal" neural network structure essentially highly relies on the assignment of an appropriate λ value to the system cost function. Similarly, the parameter w_0 in the second complexity penalty term also needs to be carefully selected. It is worth mentioning that all these subjective parameters will have non-negligible influence on the final selection of a balanced neural network structure, whereas their specific determinations are usually based on the science of experience.

In this study, we would like to testify the feasibility of incorporating certain level of autonomous exploration of the most suitable/efficient neural network structure, as well as to demonstrate a self-optimizing neural network platform capable of heuristically pruning the redundant neurons in the network, based on the highly flexible and adaptable hardware infrastructure supported by ARANN. As for further investigations and studies on the more sophisticated, convincible neural optimization criteria and methodologies, it is still a hot research topic beyond the scope of this study.

7.0 CONCLUSIONS

7.1 THESIS SUMMARY

The increasingly shrinking electronic technology and the compound complexity in modern electronic systems have resulted in substantial increases in the numbers of both hard and soft errors. It is therefore imperative that system designers build robust fault-tolerance into computational circuits, capable of detecting and recovering the damages causing the system to process improperly. Recently, the concept of *autonomous reconfigurability (AR)* has emerged to be of great interest to the whole society, which refers to a system's ability to change its structure and operations or both in response to unexpected events.

Artificial neural network (ANN), an established bio-inspired computing paradigm, has proved very effective in a variety of real-world problems and been particularly investigated for various emerging biomedical applications. Accordingly, many specialized portable ANN-based systems have been developed, as people become more active in monitoring their own health conditions and the remarkable development of pervasive healthcare techniques. Like all other electronic systems, these ANN-based systems are also increasingly vulnerable to both transient and permanent faults which sometimes can be catastrophic, especially for life-critical medical applications. Conventional fault-tolerant techniques applicable to ANN-based systems, including spatial-/temporal-redundancy, usually consume considerable system resources and energy, which can be prohibitive to meet the strict requirements of next-generation portable medical solutions. Moreover, their lack of dynamic adaptability makes their protection effective only against faults that can be conceived at the design stage.

Inspired by the precise, systematic, and essentially AR-based recovery mechanisms of the human Central Nervous System (CNS), we would like to develop a reliable ANN environment with self-healing and self-optimizing capabilities. Orthogonal with conventional reliability design techniques, we pursue an alternative way to augment the fault-tolerance and resilience of ANN-based hardware, leveraging the inherently homogeneous structural characteristics of neural networks. In principle, the ultimate goal is to achieve a reliable solution and at least maintain appropriate operational capabilities by making ANN system capable of adapting its structure or operations in response to unforeseen events.

In this dissertation, we propose a novel *Autonomously Reconfigurable Artificial Neural Network (ARANN)* architectural framework, which is capable of adapting ANN's structures and behaviors, both algorithmically and microarchitecturally, to react to unexpected neuron failures. With particular attention to the problems related to timely autonomous structural reconfiguration, ARANN could be incorporated with existing concurrent error detection (CED) techniques [227] to provide a comprehensive solution to fault-tolerant design of ANN systems. Instead of costly modular redundancy with voting, effective CED can be achieved by introducing non-intrusive circuits for coding schemes (e.g., AN codes [223], residue codes [224], redundant binary representation [17]) and the concurrent localization of faulty neurons can be realized by observing the results of signature-based error compression and propagation [54, 55]. Using the locations of faulty components reported by an error detector, ARANN can effectively achieve the ANN system recovery by excluding the faulty neurons from the active computation and reconfiguring the network structure in an autonomous manner.

More specifically, in a similar way as CNS's recovery process in case of an acquired brain injury, the proposed self-healing ARANN architecture can immediately adapt the system structure to disconnect the damaged neuron unit from the main network, if any error has been reported by the fault detector, and then involve new neuron units into the network to maintain the desired performance if any available neuron units are found. Otherwise, if no further neuron resources are available, the ARANN will continue its normal operation in a compromised mode caused by the slightly fewer neuron nodes contained by the current ANN system. Given the incomparable capabilities of connecting and disconnecting any physical neuron unit to/from the main network on the fly, ARANN will be able to evaluate the system cost involving both performance measure and complexity overhead, and then adaptively explore the most optimal network structure with appropriate design tradeoff.

The contributions of this dissertation research are threefold:

- First, we propose a bio-inspired ARANN architectural framework, capable of adapting ANN's structure and operations, to react to unexpected neuron failures. We demonstrate the effective and efficient self-healing and self-optimizing system adaptation methodologies on the ARANN, leveraging several architectural innovations which include the *Distributed ANN* architecture, the neuron virtualization technique with a *Decoupled Virtual-to-Physical Neuron Mapping*, and a *Dual-Layer Synchronization* mechanism to ensure accurate system recovery of the highly structured neural network systems.
- Secondly, to further reduce the added time latency and resource overhead associated with ARANN's dynamic structural reconfiguration, we present and investigate four possible design solutions for the most critical component in the ARANN — *Virtual-to-Physical Neuron Mapping*. A thorough analysis and comparison have been performed on all of them to explicitly demonstrate their specific applicabilities.
- Thirdly, we verify the ARANN using a real biomedical case study and prototype ARANN on the Virtex-5 FPGA platform. It is shown that ARANN can cover and adapt 93% chip area (neurons) with less than 1% chip overhead and $O(n)$ reconfiguration latency. A detailed performance analysis has been illustrated based on various recovery scenarios.

In summary, ARANN is an innovative architectural framework that can effectively address reliability issues of ANN-based hardware by automatically adapt their structures and operations without halting system execution and introducing considerable redundancy. It provides designers (particularly biomedical system designers) with a new class of highly integrated, reliable, portable, multi-functional neural network platforms that can achieve self-healing and self-optimization through automatic structural reconfiguration.

7.2 RESEARCH AIMS AND SOLUTIONS

The methodology proposed in this thesis consists of one main component and two major adaptation mechanisms. The main component is essentially a computational architecture which provides a low-cost reconfigurable hardware infrastructure for the structural and behavioral adaptations of the ANN system. The two major adaptation mechanisms, on the other hand, allow two different operational processes based on ARANN to achieve the self-healing and self-optimizing purposes respectively. This section summarizes how ARANN addresses each of aforementioned research aims (Section 1.2.1) effectively and efficiently.

7.2.1 Self-Healing ANN Solution

It is well known that the human brain has the most precise, sophisticated, and intelligent fault-tolerance capability and automatic recovery mechanism to react to unexpected injuries or diseases in the universe. ARANN achieves a reliable ANN hardware platform with self-healing capability by mimicking CNS's faulty reaction strategies and making ANN system capable of adapting its structure or operations in response to an unforeseen event. This strategic target essentially involves an optimal trade-off among system performance, reliability requirements, and associated costs. Instead of preparing a lot of identical redundant hardware components to fill in the vacant positions in case some components are physically damaged, ARANN would be capable of dynamically determining an optimal ANN structure and synaptic connections, as well as adaptively finding and incorporating available neuron resources to maintain the best achievable performance of the affected ANN system. Specifically, in a similar way as CNS's recovery process in case of a acquired brain injury, the proposed self-healing ARANN architecture can immediately adapt the system structure to disconnect the damaged neuron unit from the main network, if any error has been reported by the fault detector, and then involve new neuron units into the network to maintain the desired performance if any available neuron units are found. Otherwise, if no further neuron resources are available, the ARANN will continue its normal operation in a compromised mode caused by the slightly fewer neuron nodes contained by the current ANN system. One of the most profound benefits of the proposed self-healing enabled ARANN is the opportunity to help ANN system react to any unexpected harmful events in an autonomous, on-line, and efficient manner without halting system execution and introducing considerable redundancy.

To better utilize the massively parallel processing nature of neural networks and facilitate their structural adaptation, we propose a *Distributed Artificial Neural Network (DANN)* architecture, mainly featuring a lightweight topological & algorithmic controller and a mass of highly independent, autonomic, smart neuron units. Also, given the loosely coupled computations and communications enabled by DANN, we present a novel *Decoupled Virtual-to-Physical (V2P) Neuron Mapping* strategy to implement a cost-efficient system adaptation scheme. Specifically, we propose a "neuron virtualization" by abstracting away the direct connections between ANN controller and all physical neuron units, and inserting a flexible V2P neuron mapping block to determine an appropriate connection scheme between virtual and physical neuron ports, according to the desired number of neurons and the availability of individual physical neurons. With such a decoupling scheme, the real spatio-temporal connectionism for "physical neurons" is transparent to the controller that handles "virtual neurons". A detected faulty neuron can be timely removed from the neural network by changing the corresponding V2P mapping scheme to swap the faulty neuron with a spare neuron. In the mean time, a *Dual-Layer Memory Synchronization* mechanism ARANN in presented to ensure a smooth, accurate and consistent recovery of the highly intertemporal-dependent neural network systems no matter when an unexpected fault is detected.

7.2.2 Self-Optimization ANN Solution

To solve real-world problems using ANNs, it usually requires the use of highly structured networks of a rather large size. A rule of thumb for obtaining good generalization capability is to use the smallest system that will fit the data. Because a neural network with minimum size is less likely to learn the idiosyncrasies or noise in the training data, and may thus generalize better to new data. In addition, an ANN solution capable of providing reasonable performance with much less complexity and resource consumption is highly favored by future ultra-portable biomedical systems, which usually are extremely size- and power-concerned. Since there has not been any theory capable of directly determining the best size of neural networks, we should search and find an optimal network structure by comparing various potential candidates according to a certain evaluation criterion. One effective and efficient approach is so-called *network pruning.* It starts with a rather large neural network with sufficient neuron units for the given application, and then some inactive neurons will be gradually removed or certain synaptic weights will be eliminated in a selective and orderly fashion. This key idea is to iteratively evaluate the trade-off between the training accuracy and the structural complexity of ANN systems and then select the optimal structure providing reasonable accuracy with the least design complexity.

Although such type of optimization strategies has been extensively studied in software implementations of neural networks, there has not been any neural hardware capable of dynamically optimizing its structure and providing efficient solutions for different applications, because most neural hardware were developed for certain applications only and they are reluctant to evolve into a more efficient shape. Moreover, ANN-based hardware is expected to fit different applications in a more power-efficient manner. One possible solution to achieve this goal is to make ANN adaptable and reconfigurable and thus determine the system structure according to specific requirements and design trade-offs between performance and complexity. Leveraging the reconfigurable and adaptable architectural infrastructure provided by ARANN, we incorporated the concept of neural network pruning into ARANN and proposed a *Self-Optimizing Artificial Neural Network (SOANN)*, making use of ARANN's incomparable capabilities of connecting and disconnecting any physical neuron unit to/from the main network on the fly. Instead of determining an "optimal" structure for one certain application by the off-line analysis, the ARANN architecture will be able to evaluate the system cost involving both performance measure and complexity overhead, and then adaptively explore the most optimal network structure with the appropriate performance tradeoff. In summary, the proposed ARANN-based self-optimization approach is capable of helping users further shape the structure of neural networks and remove unnecessary (or "redundant") neurons which have little or no influence on the overall network performance.

7.2.3 Low-Cost System Adaptation

Although ARANN has proved to be effective for the self-healing and self-optimizing system adaptations, one major concern is about the costs associated with such reconfiguration efforts. It is shown that the V2P Neuron Mapper is one of the most critical components within this ARANN architecture and also the major element which introduces extra time and space overhead to the ANN system. Considering the possibility that the electronic reliability issues will become increasingly severe and the exponentially growing needs of more versatile, easily configured ANN hardware, it is highly desired to design and implement a fast, flexible, accurate, and resource-efficient V2P mapping block which can be integrated into our ARANN architecture. In this thesis, we explore several different V2P mapping design solutions from various perspectives and then analyze their specific characteristics (i.e., performance, implementation efficiency, and potential overhead) and applicabilities to pursue the lowest time and space overhead associated with the demonstrated autonomous reconfiguration capability. We propose four V2P design strategies: 1) Adaptive Physical Neuron Allocation ("V2P Mapper"), 2) Cache-Accelerated Adaptive Physical Neuron Allocation ("V2P Mapper w/ Cache"), 3) Virtual-to-Physical Neuron Mapping Memory ("V2P Memory"), and 4) Mask-Based Virtual-to-Physical Neuron Mapping Memory ("Mask-based V2P Memory"). According to thorough comparison of results, it is clearly shown that there is not a perfect design choice and all these four design strategies have distinct characteristics in design complexity, resource requirement, time overhead, and applicability to various scales of problems and thus have their own advantages and limitations.

7.3 FUTURE WORK

This thesis aims to advance the latest research efforts on robust, fault-tolerant complex systems and devices intensively required in biomedical applications, as well as to help filling the gap between current increasingly demands on autonomously, noninvasive reconfigurable bio-inspired computing techniques and their respective implementation in current commercial devices. To fulfill these objectives, this thesis involves several topics, and consequently, the work in this thesis could be continued and extended in a variety of directions.

Firstly, enhancements could be made to the current architecture, design, use and evaluation of artificial neural networks. In particular, this thesis details the implementation of an autonomously reconfigurable ANN framework based on Multilayer Perceptron (MLP), because it is so far one of the most frequently used variants of artificial neural networks. This could be extended by introducing the proposed methodology framework into other popular ANN variants, such as Support Vector Machines (SVMs), Self-Organizing Machines (SOMs), Hopfield neural networks, etc. These may offer higher performance than the current MLP structure used in this study.

The second area of future exploration is in the platform-level reconfigurability. Partial Dynamic Reconfiguration (PDR) is an emerging feature supported by modern FPGAs allowing specific regions of an FPGA to be reconfigured on the fly, hence introducing the possibility of time-sharing the available hardware resources for executing multiple needs, hardware limitations, and Quality-of-Service (QoS) requirements (power consumption, performance, execution time, etc.). If the proposed architectural reconfigurability can be combined with the reconfigurable capability supported by the hardware platform, the overall ANN systematic adaptability can be synergistically augmented to meet distinct performance and reliability demands, as well as to more intelligent neural network systems. For instance, in this study, we have primarily investigated the multi-layer perceptron neural network, whose topological structure and training methodology were determined and managed by the controller module. However, given the partial reconfiguration capability supported by FPGAs, we can design and implement many different versions of ANN controller representing distinct topological structures and learning strategies, as well as then dynamically select and load an appropriate controller module according to specific characteristics requirements of the desired application. In this case, the reconfiguration scale of ANN systems has been extended from the intra-structural adaptation within a specific neural topology to a much broader topological adaptation, which will offer compounded benefits of remarkable flexibility and reliability to ANN users.

There is a perhaps more interesting proposition. With the rapid shrinking of transistor sizes, the latest FPGAs have been able to accommodate relatively complex systems. However, the space constraint is still one of the most challenging issues to deploy a more complicated neural network on the FPGA due to ANN's highly involved computation and highly structured topological network connections, although current mainstream FPGAs has been able to accommodate some dozens of neuron units. Therefore, it might be of great interest if we could adapt the proposed autonomously reconfigurable architecture on a chip to a multi-chip environment (e.g., multi-FPGAs, multi-core chips, or chip multiprocessors (CMPs)) to further implement system-level reconfiguration which might bring more attractive performance benefits. Also, it is expected that multi-chip platforms can be able to offer enough computational capability to meet today's high performance computing demands on large scale neuromorphic simulation .

A final area of suggested further research is in various optimization approaches of neural network implementations on hardware. My dissertation research only illustrates one possible implementation scheme of the MLP neural network. However, there is still a lot of room to improve the implementation efficiency. More sophisticated design strategies can be applied to the ANN implementations to further reduce the data communication latency, resource consumption of arithmetic modules, and data dependency between all of neuron units. ARANN is orthogonal to the conventional optimization strategies of ANN hardware implementations. Thus it can be expected, based on the proposed ARANN architecture and other design optimizations, more reliable and sophisticated neural network systems can be achieved to advance the ANN's effective and efficient use in a variety of mission-critical applications.

BIBLIOGRAPHY

[1] M. Aberbour and H. Mehrez. Architecture and design methodology of the RBF-DDA neural network. In *Proceedings of the IEEE International Symposium on Circuits Systems*, volume 3, pages 199–202, 1998.

[2] D. Abramson, K. Smith, P. Logothetis, and D. Duke. FPGA based implementation of a Hopfield neural network for solving constraint satisfaction problems. In *Proceedings of the 24th EUROMICRO Conference*, volume 2, pages 228–229, 1998.

[3] P. Abusaidi, M. Klein, and B. Philofsky. *Virtex-5 FPGA System Power Design Considerations*. Xilinx, Inc., San Jose, CA, WP285 (v1.0) edition, February 2008.

[4] A. Ahmadi, M. H. Sargolzaie, S. M. Fakhraie, C. Lucas, and S. Vakili. A low-cost fault-tolerant approach for hardware implementation of artificial neural networks. In *Proceedings of the International Conference on Computer Engineering and Technology (ICCET)*, volume 2, pages 93–97, 2009.

[5] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.

[6] G. Alizadeh, J. Frounchi, M. Baradaran Nia, M. H. Zarifi, and S. Asgarifar. An FPGA implementation of an artificial neural network for prediction of cetane number. In *Proceedings of the International Conference on Computer and Communication Engineering (ICCCE)*, pages 605–608, 2008.

[7] J. Altman and G. D. Das. Autoradiographic and histological evidence of postnatal hippocampal neurogenesis in rats. *Journal of Comparative Neurobiology*, 124(3):319–335, 1965.

[8] A. Alvarez-Buylla and J. M. Garcia-Verdugo. Neurogenesis in adult subventricular zone. *Journal of Neuroscience*, 22(3):629–634, 2002.

[9] F. Amirouche and C. G. L. Espina. Application of neural networks to prosthesis fitting and balancing in joints. United States Patent US20070233267, October 2007.

[10] L. Anghel and M. Nicolaidis. Cost reduction and evaluation of temporary faults detecting technique. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 591–598, 2000.

[11] K. Appiah, A. Hunter, H. Meng, S. Yue, M. Hobden, N. Priestley, P. Hobden, and C. Pettit. A binary self-organizing map and its FPGA implementation. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 164–171, 2009.

[12] B. Arad and A. El-Amawy. Robust fault tolerant training of feedforward neural networks. In *Proceedings of the 37th Midwest Symposium on Circuits and Systems*, volume 1, pages 539–544, 1994.

[13] M. F. Bear, B. W. Connors, and M. A. Paradiso. *Neuroscience: Exploring the Brain.* Lippincott Williams & Wilkins, Baltimore, MD, third edition, 2006.

[14] M. J. Beauchamp, S. Hauck, K. D. Underwood, and K. Scott Hemmert. Architectural modifications to enhance the floating-point performance of FPGAs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(2):177–187, 2008.

[15] R. Begg, J. Kamruzzaman, and R. Sarkar. *Neural Networks in Healthcare: Potential and Challenges.* Idea Group Inc., Hershey, PA, 2006.

[16] S. Bellis, K. M. Razeeb, C. Saha, K. Delaney, C. O'Mathuna, A. Pounds-Cornish, G. de Souza, M. Colley, H. Hagras, G. Clarke, V. Callaghan, C. Argyropoulos, C. Karistianos, and G. Nikiforidis. FPGA implementation of spiking neural networks - an initial step towards building tangible collaborative autonomous agents. In *Proceedings of the Proceedings of International Conference on Field-Programmable Technology (FPT)*, pages 449–452, 2004.

[17] S. Bettola and V. Piuri. High performance fault-tolerant digital neural networks. *IEEE Transactions on Computers*, 47(3):357–363, 1998.

[18] J.-L. Beuchat, J.-O. Haenni, and E. Sanchez. *Parallel and Distributed Processing*, volume 1388 of *Lecture Notes in Computer Science*, chapter Hardware Reconfigurable Neural Networks, pages 91–98. Springer, Berlin/Heidelberg, 1998.

[19] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Inc., New York, NY, 1st edition, 1995.

[20] F. Blayo and P. Hurat. A reconfigurable WSI neural network. In *Proceedings of the 1st International Conference on Wafer Scale Integration*, pages 141–150, 1989.

[21] G. Bolt, J. Austin, and G. Morgan. Fault tolerant multi-layer perceptron networks. Technical Report YCS 180, University of York, York, U. K., July 1992.

[22] N. M. Botros and M. Abdul-Aziz. Hardware implementation of an artificial neural network. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1252–1257, 1993.

[23] N. M. Botros and M. Abdul-Aziz. Hardware implementation of an artificial neural network using field programmable gate arrays (FPGA's). *IEEE Transactions on Industrial Electronics*, 41(6):665–667, 1994.

[24] M. Bougataya, A. Lakhsasi, R. Norman, R. Prytula, Y. Blaquière, and Y. Savaria. Steady state thermal analysis of a reconfigurable wafer-scale circuit board. In *Proceedings of the Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 411–416, 2008.

[25] M. Bougataya, A. Lakhsasi, Y. Savaria, and D. Massicotte. Mixed fluid-heat transfer approach for VLSI steady state thermal analysis. In *Proceedings of the Canadian Conference on Electrical and Computer Engineering (CCECE)*, volume 1, pages 403–407, 2002.

[26] J. M. B. Braman, R. M. Murray, and D. A. Wagner. Safety verification of a fault tolerant reconfigurable autonomous goal-based robotic control system. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 853–858, 2007.

[27] L. Breveglieri and V. Piuri. Error detection in digital neural networks: An algorithm-based approach for inner product protection. In *Proceedings of the SPIE Conference — Advance Signal Processing*, pages 809–820, 1994.

[28] J. E. Brewer, A. M. Donohoo, and K. F. Olson. Automatic external defibrillator having a ventricular fibrillation detector. United States Patent US6263238, July 2001.

[29] N. Bu, M. Okamoto, and T. Tsuji. A hybrid motion classification approach for EMG-based human-robot interfaces using Bayesian and neural networks. *IEEE Transactions on Robotics*, 25(3):502–511, 2009.

[30] S. Ramón Y Cajal. *Degeneratin and Regeneration of the Nervous System.* Oxford University Press, New York, NY, 1928.

[31] H. Cecotti and A. Cräser. Neural network pruning for feature selection application to a P300 brain-computer interface. In *Proceedings of the European Symposium on Artificial Neural Networks — Advances in Computational Intelligence and Learning*, pages 473–478, 2009.

[32] C.-H. Chang, M. Shibu, and R. Xiao. Self organizing feature map for color quantization on FPGA. In Amos R. Omondi and Jagath C. Rajapakse, editors, *FPGA Implementations of Neural Networks*, pages 225–245. Springer, United States, 2006.

[33] C.-L. Chang, Z. Jin, H-C Chang, and A C. Cheng. From neuromuscular activation to end-point locomotion: An artificial neural network-based technique for neural prostheses. *Journal of Biomechanics*, 42(8):982–988, 2009.

[34] C.-L. Chang, Z. Jin, and A. C. Cheng. Predicting end-point locomotion from neuromuscular activities of people with Spina Bifida: A self-organizing and adaptive technique for future implantable and non-invasive neural prostheses. In *Proceedings of the 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 4203–4207, 2008.

[35] C.-L. Chang, M. Kubo, U. Buzzi, and B. D. Ulrich. Early changes in muscle activation patterns of toddlers during walking. *Infant Behavior and Development*, 29(2):175–188, 2006.

[36] C.-L. Chang and B. D. Ulrich. Lateral stabilization improves walking in people with myelomeningocele. *Journal of Biomechanics*, 41(6):1317–1323, 2008.

[37] Y. Chauvin. A back-propagation algorithm with optimal use of hidden units. In *Advances in Neural Information Processing Systems 1*, pages 519–526, San Francisco, CA, 1989. Morgan Kaufmann Publishers Inc.

[38] C.-H. Chen, L.-C. Chu, and D. G. Saab. Reconfigurable fault tolerant neural network. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, volume 2, pages 547–552, 1992.

[39] G. Cheron, A. M. Cebolla, A. Bengoetxea, F. Leurs, and B. Dan. Recognition of the physiological actions of the triphasic EMG pattern by a dynamic recurrent neural network. *Neuroscience Letters*, 414(2):192–196, 2007.

[40] G. Cheron, F. Leurs, A. Bengoetxea, J. P. Draye, M. Destrée, and B. Dan. A dynamic recurrent neural network for multiple muscles electromyographic mapping to elevation angles of the lower limb in human locomotion. *Journal of Neuroscience Methods*, 129(2):95–104, 2003.

[41] R. T. Chien and J. H. Se. Error correction in high-speed arithmetic. *IEEE Transactions on Computers*, C-21(5):433–438, 1972.

[42] C.-T. Chin, K. Mehrotra, C. K. Mohan, and S. Rankat. Training techniques to obtain fault-tolerant neural networks. In *Proceedings of the Twenty-Fourth International Symposium on Fault-Tolerant Computing (FTCS)*, pages 360–369, 1994.

[43] C. Christodoulou and M. Georgiopoulos. *Applications of Neural Networks in Electromagnetics.* Artech House, Inc., Norwood, MA, 2001.

[44] L.-C. Chu and B. W. Wah. Fault tolerant neural networks with hybrid redundancy. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, volume 2, pages 639–649, 1990.

[45] R. K. Chun. *Fault Tolerance Characteristics of Neural Networks*. PhD thesis, University of California, Los Angeles, Los Angeles, CA, 1989.

[46] R. K. Chun and L. P. McNamee. Immunization of neural networks against hardware faults. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, volume 1, pages 714–718, 1990.

[47] W.-Y. Chung, C.-L. Yau, K.-S. Shin, and R. Myllyla. A cell phone based health monitoring system with self analysis processor using wireless sensor network technology. In *Proceedings of the International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 3705–3708, 2007.

[48] R. D. Clay and C. H. Séquin. Fault tolerance training improves generalization and robustness. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, volume 1, pages 769–774, 1992.

[49] M. E. Cohen and D. L. Hudson. Inclusion of ECG and EEG analysis in neural network models. In *Proceedings of the 23rd International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, volume 2, pages 1621–1624, 2001.

[50] C. E. Cox and W. E. Blanz. GANGLION — a fast hardware implementation of a connectionist classifier. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 6.5/1–6.5/4, 1991.

[51] C. E. Cox and W. E. Blanz. GANGLION — a fast field-programmable gate array implementation of a connectionist classifier. *IEEE Journal of Solid-State Circuits*, 27(3):288–299, 1992.

[52] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

[53] M. Darnell, B. K. Honary, and F. Zolghardr. Embedded coding technique: Principles and theoretical studies. *IEEE Proceedings F: Radar and Signal Processing*, 135(1):43–50, 1988.

[54] S. Demidenko and V. Piuri. On-line testing in digital neural networks. In *Proceedings of the 5th Asian Test Symposium (ATS)*, page 295, 1996.

[55] S. Demidenko and V. Piuri. Concurrent diagnosis in digital implementations of neural networks. *Neurocomputing*, 48(1-4):879–903, 2002.

[56] S. Demont-Guignard, P. Benquet, U. Gerber, and F. Wendling. Analysis of intracerebral EEG recordings of epileptic spikes: Insights from a neural network model. *IEEE Transactions on Biomedical Engineering*, 56(12):2782–2795, 2009.

[57] D. Deodhare, M. Vidyasagar, and S. S. Keerthi. Synthesis of fault-tolerant feedforward neural networks using minimax optimization. *IEEE Transactions on Neural Networks*, 9(5):891–900, 1998.

[58] B. S. Dhillon. *Medical Device Reliability and Associated Areas.* CRC Press LLC, Boca Raton, FL, 2000.

[59] B. S. Dhillon. Medical equipment reliability. In J. Mira, R. Moreno-Diaz, and J. Cabestany, editors, *Applied Reliability and Quality*, Springer Series in Reliability Engineering, pages 79–96. Springer, London, Britain, 2007.

[60] F. M. Dias, A. Antunes, and A. M. Mota. Artificial neural networks: A review of commercial hardware. *Engineering Applications of Artificial Intelligence*, 17(8):945–952, 2004.

[61] B. W. Dickinson. Structured neural networks for fault tolerant performance. In *Proceedings of the 29th IEEE Conference on Decision and Control*, volume 5, pages 2741–2743, 1990.

[62] S. Draghici. On the computational power of limited precision weights neural networks in classification problems: How to calculate the weight range so that a solution will exist. In J. Mira and J. V. Sanchez-Andres, editors, *Foundations and Tools for Neural Modeling*, volume 1606 of *Lecture Notes in Computer Science*, pages 401–412. Springer, Berlin/Heidelberg, Germany, 1999.

[63] S. Draghici. On the capabilities of neural networks using limited precision weights. *Neural Networks*, 15(3):395–414, 2002.

[64] S. Draghici and I. K. Sethi. On the possibilities of the limited precision weights neural networks in classification problems. In J. Mira, R. Moreno-Diaz, and J. Cabestany, editors, *Biological and Artificial Computation: From Neuroscience to Technology*, volume 1240 of *Lecture Notes in Computer Science*, pages 753–762. Springer, Berlin/Heidelberg, Germany, 1997.

[65] R. Dybowski and V. Gant. *Clinical Applications of Artificial Neural Networks.* Cambridge University Press, New York, NY, 1st edition, 2007.

[66] G. P. K. Economou, E. P. Mariatos, N. M. Economopoulos, D. Lymberopoulos, and C. E. Goutis. FPGA implementation of artificial neural networks: An application on medical expert systems. In *Proceedings of the 4th International Conference on Microelectronics for Neural Networks and Fuzzy Systems*, pages 287–293, 1994.

[67] P. E. Edwards. Arithmetic with binary numbers. Lecture Notes for "Introduction to Computer Architecture", Imperial College London, UK, 2006.

[68] H. M. El-Bakry and N. Mastorakis. A simple design and implementation of reconfigurable neural networks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 744–750, 2009.

[69] J. G. Eldredge and B. L. Hutchings. Density enhancement of a neural network using FPGAs and run-time reconfiguration. In *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines (FCCM)*, pages 180–188, 1994.

[70] J. G. Eldredge and B. L. Hutchings. RRANN: The run-time reconfiguration artificial neural network. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 77–80, 1994.

[71] H. Elsimary, S. Mashali, and S. Shaheen. A method for training feed forward neural network to be fault tolerant. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 436–441, 1993.

[72] P. S. Eriksson, E. Perfilieva, T. Björk-Eriksson, A.-M. Alborn, C. Nordborg, D. A. Peterson, and F. H. Gage. Neurogenesis in the adult human hippocampus. *Nature Medicine*, 4:1313–1317, 1998.

[73] C. T. Ewe, P. Y. K. Cheung, and G. A. Constaninides. Error modelling of dual fixed-point arithmetic and its application in field programmable logic. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pages 124–129, 2005.

[74] K. Falconer and D. A. Winter. Quantitative assessment of co-contraction at the ankle joint in walking. *Electroencephalography and Clinical Neurophysiology*, 25(2-3):135–149, 1985.

[75] J. W. Fawcett and R. A. Asher. The glial scar and central nervous system repair. *Brain Research Bulletin*, 49(6):377–391, 1999.

[76] S. Ferrante, A. Pedrocchi, M. Iannò, E. De Momi, M. Ferrarin, and G. Ferrigno. Functional electrical stimulation controlled by artificial neural networks: Pilot experiments with simple movements are promising for rehabilitation applications. *Functional Neurology*, 19(4):243–252, 2004.

[77] N. Fisekovic and D. B. Popovic. New controller for functional electrical stimulation systems. *Medical Engineering & Physics*, 23(6):391–399, 2001.

[78] International Technology Roadmap for Semiconductors (ITRS). International technology roadmap for semiconductors report 2009 edition, 2009.

[79] J. Frounchi, G. Karimian, and A. Keshtkar. An artificial neural network hardware for bladder cancer. *European Journal of Scientific Research*, 27(1):46–55, 2009.

[80] O. Fujita. Statistical estimation of the number of hidden units for feedforward neural networks. *Neural Networks*, 11(5):851–859, 1998.

[81] J. A. Fulcher. A comparative review of commercial ann simulators. *Computer Standards & Interfaces*, 16:241–251, 1994.

[82] A. Golander, S. Weiss, and R. Ronen. DDMR: Dynamic and scalable dual modular redundancy with short validation intervals. *IEEE Computer Architecture Letters*, 7(2):65–68, 2008.

[83] R. M. Goodman and M. Sayano. The reliability of semiconductor RAM memories with on-chip error-correction coding. *IEEE Transactions on Information Theory*, 37(3):884–896, 1991.

[84] M. Gorgoń and M. Wrzesiński. Neural network implementation in reprogrammable FPGA devices — an example of MLP. In L. Rutkowski and J. Kacprzyk, editors, *Artificial Intelligence and Soft Computing*, volume 4029 of *Lecture Notes in Computer Science*, pages 19–28. Springer, Berlin/Heidelberg, Germany, 2006.

[85] K. Goser, U. Hilleringmann, and U. Rückert. Application and implementation of neural networks in microelectronics. In *Artificial Neural Networks*, volume 540 of *Lecture Notes in Computer Science*, pages 243–259. Springer, Berlin/Heidelberg, Germany, 1991.

[86] H. P. Graf and D. Henderson. A reconfigurable CMOS neural network. In *Proceedings of the International Solid-State Circuits Conference (ISSCC)*, pages 144–145, 1990.

[87] H. P. Graf and L. D. Jackel. Analog electronic neural network circuits. *IEEE Circuits Devices Magazine*, 5(4):44–49, 1989.

[88] D. Graupe and H. Kordylewski. Artificial neural network control of FES in paraplegics for patient responsive ambulation. *IEEE Transactions on Biomedical Engineering*, 42(7):699–707, 1995.

[89] D. E. Grierson and P. Hajela. *Emergent Computing Methods in Engineering Design: Applications of Genetic Algorithms and Neural Networks.* Springer-Verlag New York, LLC, New York, NY, 1st edition, 1996.

[90] M. Gschwind, V. Salapura, and O. Maischberger. RAN2SOM: A reconfigurable neural network architecture based on bit stream arithmetic. In *Proceedings of the 4th International Conference on Microelectronics for Neural Networks and Fuzzy Systems*, pages 294–300, 1994.

[91] K. Gurney. *Learning in Networks of Structured Hypercubes.* PhD thesis, Brunel University, England, United Kingdom, 1989.

[92] J.-O. Haenni, J.-L. Beuchat, and E. Sanchez. RENCO: A reconfigurable network computer. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, pages 288–289, 1998.

[93] M. T. Hagan, H. B. Demuth, and M. H. Beale. *Neural Network Design.* PWS Publishing Co., Boston, MA, 1997.

[94] M. Hagiwara. Removal of hidden units and weights for back propagation networks. In *Proceedings of the International Joint Conference on Neural Networks*, volume 1, pages 351–354, 1993.

[95] M. E. Hahn, A. M. Farley, V. Lin, and L.-S. Chou. Neural network estimation of balance control during locomotion. *Journal of Biomechanics*, 38(4):717–724, 2005.

[96] N. C. Hammadi, T. Ohmameuda, K. Kaneko, and H. Ito. Fault tolerant constructive algorithm for feedforward neural networks. In *Proceedings of the Pacific Rim International Symposium on Fault-Tolerant Systems*, pages 215–220, 1997.

[97] D. Hammerstrom. A highly parallel digital architecture for neural network emulation. In J. G. Delgado-Frias and W. R. Moore, editors, *VLSI for Artificial Intelligence and Neural Networks*, pages 357–366. Plenum Press, New York, 1991.

[98] H. Han and J. Qiao. A novel pruning algorithm for self-organizing neural network. In *Proceedings of the International Joint Conference on Neural Networks (ICJNN)*, pages 1245–1250, 2009.

[99] J. Harkin, F. Morgan, S. Hall, P. Dudek, T. Dowrick, and L. McDaid. Reconfigurable platforms and the challenges for large-scale implementations of spiking neural networks. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pages 483–486, 2008.

[100] T. Haruhiko, N. Ayumi, K. Hidehiko, and H. Terumine. Dynamic construction of fault tolerant multi-layer neural networks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, volume 2, pages 995–999, 2005.

[101] T. Haruhiko, K. Hidehiko, and H. Terumine. Partially weight minimization approach for fault tolerant multilayer neural networks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, volume 2, pages 1092–1096, 2002.

[102] M. A. Hasan, M. I. Ibrahimy, M. B. I. Reaz, M. J. Uddin, and M. S. Hussain. VHDL modeling of FECG extraction from the composite abdominal ECG using artificial intelligence. In *Proceedings of the IEEE International Conference on Industrial Technology*, pages 1–5, 2009.

[103] S. Haykin. *Neural Network: A Comprehensive Foundation.* Pearson Education, Inc., Upper Saddle River, NJ, second edition, 1998.

[104] D. O. Hebb. *The Organization of Behavior: A Neuropsychological Theory.* Wiley-Interscience, New York, NY, 1949.

[105] H. H. Hellmich and H. Klar. A FPGA based simulation acceleration platform for spiking neural networks. In *Proceedings of the 47th Midwest Symposium on Circuits and Systems (MWSCAS)*, volume 2, pages 389–392, 2004.

[106] J. G. Hincapie and R. F. Kirsch. Feasibility of EMG-based neural network controller for an upper extremity neuroprosthesis. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 17(1):80–90, 2007.

[107] G. E. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40(1-3):185–234, 1989.

[108] J. L. Holt and T. E. Baker. Back propagation simulations using limited precision calculations. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, volume 2, pages 121–126, 1991.

[109] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, volume 79, pages 2554–2558, 1982.

[110] T. Horita, T. Murata, and I. Takanami. A multiple-weight-and-neuron-fault tolerant digital multilayer neural network. In *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 554–562, 2006.

[111] T. Horita and I. Takanami. Novel learning algorithms which make multilayer neural networks multiple-weight-and-neuron-fault tolerant. In *Proceedings of the International Conference on Neural Information Processing (ICONIP)*, pages 564–569, 2005.

[112] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[113] K. Hornik. Some new results on neural network approximation. *Neural Networks*, 6(9):1069–1072, 1993.

[114] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[115] W. S. Hsieh and B. Y. Sher. Fault tolerant capability of multi-layer perceptron neural network. In *Proceedings of the 20th EUROMICRO Conference on System Architecture and Integration*, pages 644–650, 1994.

[116] Y.-M. Hsu, V. Piuri, and E. E. Swartzlander Jr. Time-redundant multiple computation for fault-tolerant digital neural networks. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, volume 2, pages 977–980, 1995.

[117] Y.-M. Hsu, E. E. Swartzlander Jr., and V. Piuri. Recomputing by operand exchanging: A time-redundancy approach for fault-tolerant neural networks. In *Proceedings of the International Conference on Application-Specific Array Processors (ASAP)*, pages 54–65, 1995.

[118] H. Hu, J. Huang, J. Xing, and W. Wang. Key issues of FPGA implementation of neural networks. In *Proceedings of the International Symposium on Intelligent Information Technology Application (IITA)*, volume 3, pages 259–263, 2008.

[119] Y. H. Hu, S. Palreddy, and W. J. Tompkins. A patient-adaptable ECG beat classifier using a mixture of experts approach. *IEEE Transactions on Biomedical Engineering*, 44(9):891–900, 1997.

[120] Y. H. Hu, W. J. Tompkins, and Q. Xue. Artificial neural network for ECG arrhythmia monitoring. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, volume 2, pages 987–992, 1992.

[121] D. L. Hudson and M. E. Cohen. *Neural Networks and Artificial Intelligence for Biomedical Engineering.* Wiley-IEEE Press, New York, NY, 1999.

[122] D. L. Hudson, M. E. Cohen, W. Meecham, and M. Kramer. Inclusion of signal analysis in a hybrid medical decision support system. *Methods of Information in Medicine*, 43(1):79–82, 2004.

[123] IEEE Computer Society. *IEEE Standard for Binary Floating-Point Arithmetic.* IEEE Standards Board, New York, NY, IEEE Std 754-1985 edition, March 1985.

[124] IEEE Computer Society. *IEEE Standard for Floating-Point Arithmetic.* IEEE-SA Standards Board, New York, NY, IEEE Std 754-2008 edition, June 2008.

[125] M. Ishikawa. A structural learning algorithm with forgetting of link weights. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, volume 2, page 626, 1989.

[126] M. M. Islam, X. Yao, and K. Murase. A constructive algorithm for training cooperative neural network ensembles. *IEEE Transactions on Neural Networks*, 14(4):820–834, 2003.

[127] H. Ito and T. Yagi. Fault tolerant design using error correcting code for multilayer neural networks. In *Proceedings of the IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems (DFTVS)*, pages 177–184, 1994.

[128] K. Iwasa, M. Kugler, S. Kuroyanagi, and A. Iwata. A sound localization and recognition system using pulsed neural networks on fpga. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 902–907, 2007.

[129] E. M. Izhikevich. Polychronization: Computation with spikes. *Neural Computation*, 18(2):245–282, 2006.

[130] A. Jahnke, U. Roth, and H. Klar. A SIMD/dataflow architecture for a neurocomputer for spike-processing neural networks (NESPINN). In *Proceedings of the 5th International Conference on Microelectronics for Neural Networks*, pages 232–237, 1996.

[131] W. James. *The Principles of Psychology.* Holt, New York, NY, 1890.

[132] R. Jané, J. A. Fiz, J. Solà-Soler, S. Blanch, P. Artís, and J. Morera. Automatic snoring signal analysis in sleep studies. In *Proceedings of the International Conference of the*

IEEE Engineering in Medicine and Biology Society (EMBC), volume 1, pages 366–369, 2003.

[133] C. Ji, R. R. Snapp, and D. Psaltis. Generalizing smoothness constraints from discrete samples. *Neural Computation*, 2(2):188–197, 1990.

[134] W. Jiang and S. G. Kong. Block-based neural networks for personalized ECG signal classification. *IEEE Transactions on Neural Networks*, 18(6):1750–1761, 2007.

[135] Z. Jin, Y. Sun, and A. C. Cheng. Predicting cardiovascular disease from real-time electrocardiographic monitoring: An adaptive machine learning approach on a cell phone. In *Proceedings of the International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 6889–6892, 2009.

[136] S. Jonić, T. Janković, V. Gajić, and D. Popović. Three machine learning techniques for automatic determination of rules to control locomotion. *IEEE Transactions on Biomedical Engineering*, 46(3):300–310, 1999.

[137] S. Jung and S. S. Kim. Hardware implementation of a real-time neural network controller with a DSP and an FPGA for nonlinear systems. *IEEE Transactions on Industrial Electronics*, 54(1):265–271, 2007.

[138] N. Kamiura, T. Isokawa, and N. Matsui. Learning based on fault injection and weight restriction for fault-tolerant hopfield neural networks. In *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 339–346, 2004.

[139] J. Kamruzzaman, R. K. Begg, and R. A. Sarker. *Artificial Neural Networks in Finance and Manufacturing*. Idea Group Publishing, Hershey, PA, illustrated edition, 2006.

[140] N. B. Karayiannis, A. Mukherjee, J. R. Glover, P. Y. Ktonas, J. D. Frost Jr., R. A. Hrachovy, and E. M. Mizrahi. Detection of pseudosinusoidal epileptic seizure segments in the neonatal EEG by cascading a rule-based algorithm with a neural network. *IEEE Transactions on Biomedical Engineering*, 53(4):633–641, 2006.

[141] E. D. Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE Transactions on Neural Networks*, 1(2):239–242, 1990.

[142] M. Khare and S. M. S. Nagendra. *Artificial Neural Networks in Vehicular Pollution Modelling (Studies in Computational Intelligence)*. Springer-Verlag New York, LLC, New York, NY, 1st edition, 2006.

[143] J. H. Kim, C. Lursinsap, and S. Park. Fault-tolerant artificial neural networks. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, volume 2, page 951, 1991.

[144] J.-S. Kim and S. Jung. Evaluation of embedded RBF neural chip with back-propagation algorithm for pattern recognition tasks. In *Proceedings of the IEEE International Conference on Industrial Informatics (INDIN)*, pages 1110–1115, 2008.

[145] J.-S. Kim and S. Jung. Implementation of the RBF neural chip with the on-line learning back-propagation algorithm. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, pages 377–383, 2008.

[146] S. S. Kim and S. Jung. Hardware implementation of a real time neural network controller with a DSP and an FPGA. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, volume 5, pages 4639–4644, 2004.

[147] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, 1982.

[148] T. Kohonen. *Self-Organizing Maps*. Springer Series in Information Sciences. Springer, New York, NY, third edition, 2001.

[149] H. Kordylewski, D. Graupe, and K. Liu. A novel large-memory neural network as an aid in medical diagnosis applications. *IEEE Transactions on Information Technology in Biomedicine*, 5(3):202–209, 2001.

[150] M. Krid, A. Dammak, and D. Sellami Masmoudi. FPGA implementation of programmable pulse mode neural network with on chip learning for signature application. In *Proceedings of the 13th International Conference on Electronics, Circuits and Systems (ICECS)*, pages 942–945, 2006.

[151] M. Krid, D. Sellami Masmoudi, and M. Chtourou. Hardware implementation of BFNN and RBFNN in FPGA technology: Quantization issues. In *Proceedings of the 13th International Conference on Electronics, Circuits and Systems (ICECS)*, pages 1–4, 2005.

[152] M. Krips, T. Lammert, and A. Kummert. FPGA implementation of a neural network for a real-time hand tracking system. In *Proceedings of the First International Workshop on Electronic Design, Test and Applications*, pages 313–317, 2002.

[153] S. Krishnamohan and N. R. Mahapatra. Combining error masking and error detection plus recovery to combat soft errors in static CMOS circuits. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, pages 40–49, 2005.

[154] A. Krogh. What are artificial neural networks? *Nature Biotechnology*, 26(2):195–197, 2008.

[155] B. Küpper. Implantable medical device system with sensor for hemodynamic stability and method of use. United States Patent US6615083, September 2003.

[156] K. A. Kwiat. Dynamically reconfigurable fpga apparatus and method for multiprocessing and fault tolerance. United States Patent US5931959, August 1999.

[157] P. K. Lala. *Fault Tolerant & Fault Testable Hardware Design.* Prentice Hall, Englewood Cliffs, NJ, 1985.

[158] J. M. Lary and L. D. Edmonds. Prevalence of Spina Bifida at birth — United States, 1983–1990: a comparison of two surveillance systems. *MMWR CDC Surveillance Summaries*, 45(2):15–26, 1996.

[159] M. Laubach, J. Wessberg, and M. A. L. Nicolelis. Cortical ensemble activity increasingly predicts behaviour outcomes during learning of a motor task. *Nature*, 405(6786):567–571, 2000.

[160] J.-W. Lee and G.-K. Lee. Gait angle prediction for lower limb orthotics and protheses using an EMG signal and neural networks. *International Journal of Control, Automation, and Systems*, 3(2):152–158, 2005.

[161] B. J. Leiner, V. Q. Lorena, T. M. Cesar, and M. V. Lorenzo. Hardware architecture for FPGA implementation of a neural network and its application in images processing. In *Proceedings of the Electronics, Robotics and Automotive Mechanics Conference (CERMA)*, pages 405–410, 2008.

[162] Y. Li Cun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems 2*, pages 598–605, San Francisco, CA, 1990. Morgan Kaufmann Publishers Inc.

[163] Y. Liao. Neural networks in hardware: A survey. Technical report, University of California, Davis, Davis, CA, 1999.

[164] D. C. Lie, H. Song, S. A. Colamarino, G. l. Ming, and F. H. Gage. Neurogenesis in the adult brain: New strategies for central nervous system diseases. *Annual Review of Pharmacology and Toxicology*, 44:399–421, 2004.

[165] W. B. Ligon III, S. McMillan, G. Monn, K. Schoonover, F. Stivers, and K. D. Underwood. A re-evaluation of the practicality of floating-point operations on FPGAs. In *Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines (FCCM)*, pages 206–215, 1998.

[166] J. Lin, X.-G. Jin, and J.-G. Yang. A hybrid neural network model for consciousness. *Journal of Zhejiang University — SCIENCE*, 5(11):1440–1448, 2004.

[167] J.-S. Lin, K.-S. Cheng, and C.-W. Mao. A fuzzy Hopfield neural network for medical image segmentation. *IEEE Transactions on Nuclear Science*, 43(4):2389–2398, 1996.

[168] C. S. Lindsey and T. Lindblad. Review of hardware neural networks: A user's perspective. In *Proceedings of the 3rd Workshop on Neural Networks: From Biology to High Energy Physics*, Marciana Marina, Isola d'Elba, Italy, September 1994.

[169] C. S. Lindsey, T. Lindblad, G. Sekhniaidze, G. Székely, and M. Minerskjöld. Experience with the IBM ZISC036 neural network chip. In *Proceedings of the 3rd International Workshop on Software Engineering, Artificial Intelligence, and Expert Systems, for High Energy and Nuclear Physics*, 1995.

[170] P. J. G. Lisboa, E. C. Ifeachor, and P. S. Szczepaniak. *Artificial Neural Networks in Biomedicine (Perspectives in Neural Computing)*. Springer-Verlag London, London, Great Britain, 2000.

[171] D. Liu, Z. Pang, and S. R. Lloyd. A neural network method for detection of obstructive sleep apnea and narcolepsy based on pupil size and EEG. *IEEE Transactions on Neural Networks*, 19(2):308–318, 2008.

[172] G. P. Liu. *Nonlinear Identification and Control: A Neural Network Approach (Advances in Industrial Control)*. Springer-Verlag New York, LLC, New York, NY, 1st edition, 2001.

[173] Y. Liu, J. A. Starzyk, and Z. Zhu. Optimizing number of hidden neurons in neural networks. In *Proceedings of the 25th IASTED International Conference on Artificial Intelligence and Applications*, pages 121–126, 2007.

[174] D. Lloyd-Jones, R. J. Adams, T. M. Brown, M. Carnethon, S. Dai, G. De Simone, T. B. Ferguson, E. Ford, K. Furie, C. Gillespie, A. Go, K. Greenlund, N. Haase, S. Hailpern, P. M. Ho, V. Howard, B. Kissela, S. Kittner, D. Lackland, L. Lisabeth, A. Marelli, M. M. McDermott, J. Meigs, D. Mozaffarian, M. Mussolino, G. Nichol, V. L. Roger, W. Rosamond, R. Sacco, P. Sorlie, R. Stafford, T. Thom, S. Wasserthiel-Smoller, N. D. Wong, J. Wylie-Rosett, and American Heart Association Statistics Committee and Stroke Statistics Subcommittee. Heart disease and stroke statistics 2010 update: A report from the american heart association. *Circulation*, 121(7):e46–e215, 2010.

[175] B.-L. Lu, J. Shin, and M. Ichikawa. Massively parallel classification of single-trial EEG signals using a Min-Max modular neural network. *IEEE Transactions on Biomedical Engineering*, 51(3):551–558, 2004.

[176] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgford. Enhanced architectures, design methodologies and CAD tools for dynamic reconfiguration of Xilinx FPGAs. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–6, 2006.

[177] W. Maass. Computing with spiking neurons. In W. Maass and C. M. Bishop, editors, *Pulsed Neural Networks*, pages 55–85. MIT Press, Cambridge, MA, 1999.

[178] Y. Maeda and Y. Fukuda. FPGA implementation of pulse density Hopfield neural network. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 700–704, 2007.

[179] Y. Maeda and T. Tada. FPGA implementation of a pulse density neural network using simultaneous perturbation. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, volume 3, pages 296–301, 2000.

[180] Y. Maeda and T. Tada. FPGA implementation of a pulse density neural network with learning ability using simultaneous perturbation. *IEEE Transactions on Neural Networks*, 14(3):688–695, 2003.

[181] Y. Maeda and M. Wakamura. Simultaneous perturbation learning rule for recurrent neural networks and its FPGA implementation. *IEEE Transactions on Neural Networks*, 16(6):1664–1672, 2005.

[182] T. Matuki, T. Kudo, T. Kondo, and J. Ueno. Three dimentional medical images of the lungs and brain recognized by artificial neural networks. In *Proceedings of the SICE Annual Conference*, pages 1117–1121, 2007.

[183] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1953.

[184] M. Mehrara, M. Attariyan, S. Shyam, K. Constantinides, V. Bertacco, and T. Austin. Low-cost protection for SER upsets and silicon defects. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 1146–1151, 2007.

[185] G. Minchin and A. Znknich. A design for FPGA implementation of the probabilistic neural network. In *Proceedings of the 6th International Conference on Neural Information Processing (ICONIP)*, volume 2, pages 556–559, 1999.

[186] G.-L. Ming and H. Song. Adult neurogenesis in the mammalian central nervous system. *Annual Review of Neuroscience*, 28:223–250, 2005.

[187] R. Mizuno, N. Aibe, M. Yasunaga, and I. Yoshihara. Reconfigurable architecture for probabilistic neural network system. In *Proceedings of the International Conference on Field-Programmable Technology (FPT)*, pages 367–370, 2003.

[188] P. Moerland and E. Fiesler. Neural network adaptations to hardware implementations. In E. Fiesler and R. Beale, editors, *Handbook of Neural Computation*, volume E1.2, pages 1–13. IOP Publishing Ltd and Oxford University Press, New York, 2006.

[189] G. B. Moody and R. G. Mark. The impact of the MIT-BIH arrhythmia database. *IEEE Engineering in Medicine and Biology Magazine*, 20(3):45–50, 2001.

[190] F. Moreno, J. Alarcón, R. Salvador, and T. Riesgo. Reconfigurable hardware architecture of a shape recognition system based on specialized tiny neural networks with online training. *IEEE Transactions on Industrial Electronics*, 56(8):3253–3263, 2009.

[191] P. Morgan, A. Ferguson, and H. Bolouri. Cost-performance analysis of FPGA, VLSI and WSI implementations of a RAM-based neural network. In *Proceedings of the 4th*

International Conference on Microelectronics for Neural Networks and Fuzzy Systems, pages 235–243, 1994.

[192] S. Morita, T. Kondo, and K. Ito. Estimation of forearm movement from EMG signal and application to prosthetic hand control. In *Proceedings of the nternational Conference on Robotics & Automation*, pages 3692–3697, 2001.

[193] M. Moussa, S. Areibi, and K. Nichols. On the arithmetic precision for implementating back-propagation networks on FPGA: A case study. In Amos R. Omondi and Jagath C. Rajapakse, editors, *FPGA Implementations of Neural Networks*, pages 37–61. Springer, The Netherlands, 2006.

[194] M. C. Mozer and P. Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Advances in Neural Information Processing Systems 1*, pages 107–115, San Francisco, CA, 1989. Morgan Kaufmann Publishers Inc.

[195] N. Murata, S. Yoshizawa, and S. Amari. Network information criterion — determining the number of hidden units for an artificial neural network model. *IEEE Transactions on Neural Networks*, 5(6):856–872, 1994.

[196] J. M. J. Murre. Neurosimulators. In Michael A. Arbib, editor, *Handbook of Brain Theory and Neural Networks*, pages 634–639. The MIT Press, Cambridge, MA, 1995.

[197] A. Muthuramalingam, S. Himavathi, and E. Srinivasan. Neural network implementation using FPGA: Issues and application. *International Journal of Information Technology*, 4(2):86–92, 2008.

[198] K. Nagami, K. Oguri, T. Shiozawa, H. Ito, and R. Konishi. Plastic cell architecture: Towards reconfigurable computing for general-purpose. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, pages 68–77, 1998.

[199] C. Neti, M. H. Schneider, and E. D. Young. Maximally fault-tolerant neural networks and nonlinear programming. *IEEE Transactions on Neural Networks*, 3(1):14–23, 1992.

[200] H. Ney, U. Essen, and R. Kneser. On the estimation of 'small' probabilities by leaving-one-out. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(12):1202–1212, 1995.

[201] M. Norgaard, O. Ravn, N. K. Poulsen, and L. K. Hansen. *Neural Networks for Modelling and Control of Dynamic Systems: A Practitioner's Handbook.* Springer-Verlag New York, LLC, New York, NY, 2000.

[202] M. A. Nussbaum, D. B. Chaffin, and B. J. Martin. A back-propagation neural network model of lumbar muscle recruitment during moderate static exertions. *Journal of Biomechanics*, 28(9):1015–1024, 1995.

[203] E. L. Oberstar. *Fixed-Point Representation and Fractional Math*. Oberstar Consulting, revision 1.2 edition, August 2007.

[204] A. R. Omondi, J. C. Rajapakse, and M. Bajger. FPGA neurocomputers. In Amos R. Omondi and Jagath C. Rajapakse, editors, *FPGA Implementations of Neural Networks*, pages 1–36. Springer, The Netherlands, 2006.

[205] S. Oniga, A. Tisan, D. Mic, A. Buchman, and A. Vida-Ratiu. Hand postures recognition system using artificial neural networks implemented in FPGA. In *Proceedings of the 30th International Spring Seminar on Electronics Technology*, pages 507–512, 2007.

[206] S. Oniga, A. Tisan, D. Mic, A. Buchman, and A. Vida-Ratiu. Optimizing FPGA implementation of feed-forward neural networks. In *Proceedings of the 11th International Conference on Optimization of Electrical and Electronic Equipment (OPTIM)*, pages 31–36, 2008.

[207] S. Oniga, A. Tisan, D. Mic, C. Lung, I. Orha, A. Buchman, and A. Vida-Ratiu. FPGA implementation of feed-forward neural networks for smart devices development. In *Proceedings of the International Symposium on Signals, Circuits and Systems (ISSCS)*, pages 1–4, 2009.

[208] S. Osowski and T. H. Linh. ECG beat recognition using fuzzy hybrid neural network. *IEEE Transactions on Biomedical Engineering*, 48(11):1265–1271, 2001.

[209] M. C. Patrick. Reconfigurable computing concepts for space missions: Universal modular spares. In *Proceedings of the IEEE Aerospace Conference*, pages 1–8, 2008.

[210] C. S. Pattichis, C. N. Schizas, and L. T. Middleton. Neural network models in EMG diagnosis. *IEEE Transactions on Biomedical Engineering*, 42(5):486–496, 1995.

[211] M. J. Pearson, C. Melhuish, A. G. Pipe, M. Nibouche, I. Gilhesphy, K. Gurney, and B. Mitchinson. Design and FPGA implementation of an embedded real-time biologically plausible spiking neural network processor. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pages 582–585, 2005.

[212] M. J. Pearson, A. G. Pipe, B. Mitchinson, K. Gurney, C. Melhuish, I. Gilhespy, and M. Nibouche. Implementing spiking neural networks for real-time signal-processing and control applications: A model-validated FPGA approach. *IEEE Transactions on Neural Networks*, 18(5):1472–1487, 2007.

[213] A. Pérez-Uribe and E. Sanchez. FPGA implementation of an adaptable-size neural network. In C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff, editors, *Artificial Neural Networks*, volume 1112 of *Lecture Notes in Computer Science*, pages 383–388. Springer, Berlin/Heidelberg, Germany, 1996.

[214] T. Petsche and B. W. Dickinson. Trellis codes, receptive fields, and fault tolerant, self-repairing neural networks. *IEEE Transactions on Neural Networks*, 1(2):154–166, 1990.

[215] H. Pham. Optimal cost-effective design of triple-modular-redundancy-with-spares systems. *IEEE Transactions on Reliability*, 42(3):369–374, 1993.

[216] D. S. Phatak. *Fault Tolerance of Feed-Forward Artificial Neural Nets and Synthesis of Robust Nets*. PhD thesis, University of Massachusetts, Amherst, 1994.

[217] D. S. Phatak. Fault tolerant artificial neural networks. In *Proceedings of the 5th IEEE International Dual Use Technologies and Applications Conference*, pages 193–198, 1995.

[218] D. S. Phatak and I. Koren. Fault tolerance of feedforward neural nets for classification tasks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, volume 2, pages 386–391, 1992.

[219] D. S. Phatak and I. Koren. Complete and partial fault tolerance of feedforward neural nets. *IEEE Transactions on Neural Networks*, 6(2):446–456, 1995.

[220] D. S. Phatak and E. Tchernev. Synthesis of fault tolerant neural networks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, volume 2, pages 1475–1480, 2002.

[221] V. Piuri. An algorithmic approach to concurrent error detection in artificial neural networks. In *Proceedings of the International Workshop on VLSI Signal Processing*, 1992.

[222] V. Piuri. Analysis of fault tolerance in artificial neural networks. *Journal of Parallel and Distributed Computing*, 61(1):18–48, 2001.

[223] V. Piuri, M. Sami, and R. Stefanelli. Arithmetic codes for concurrent error detection in artificial neural networks: The case of AN+B codes. In *Proceedings of the International Workshop on Defect and Fault Tolerance in VLSI Systems*, 1992.

[224] V. Piuri and M. Villa. Residue codes for concurrent error detection in artificial neural networks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 825–830, 1993.

[225] P. Plaskonos, S. Pakzad, B. Jin, and A. R. Hurson. Design of a modular chip for a reconfigurable artificial neural network. In *Proceedings of the International Conference on Developing and Managing Intelligent System Projects*, pages 55–62, 1993.

[226] M. Porrmann, U. Witkowski, H. Kalte, and U. Rückert. Implementation of self-organizing feature maps in reconfigurable hardware. In Amos R. Omondi and Jagath C. Rajapakse, editors, *FPGA Implementations of Neural Networks*, pages 247–269. Springer, United States, 2006.

[227] D. K. Pradhan. *Fault Tolerant Computing: Theory and Techniques.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1986.

[228] D. K. Pradhan. *Fault-Tolerant Computer System Design.* Prentice-Hall, Inc., Englewood Cliffs, NJ, first edition, 1996.

[229] S. D. Prentice, A. E. Patla, and D. A. Stacey. Simple artificial neural network models can generate basic muscle activity patterns for human locomotion at different speeds. *Experimental Brain Research*, 123(4):474–480, 1998.

[230] J. R. Rabuñal and J. Dorrado. *Artificial Neural Networks in Real-Life Applications.* Idea Group Inc., Hershey, PA, 2006.

[231] J. I. Raffel. Electronic implementation of neuromorphic systems. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 10.1/1–10.1/7, 1988.

[232] U. Ramacher. SYNAPSE: A neurocomputer that synthesizes neural algorithms on a parallel systolic engine. *Journal of Parallel and Distributed Computing*, 14(3):306–318, 1992.

[233] R. Reed. Pruning algorithms — a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993.

[234] I. T. Rekanos. Neural-network-based inverse-scattering technique for online microwave medical imaging. *IEEE Transactions on Magnetics*, 38(2):1061–1064, 2002.

[235] C. Ricci and S. Palreddy. Neural network based learning engine to adapt therapies. United States Patent US20070156187, July 2007.

[236] A. F. Rickards. Physiologically adaptive cardiac packmaker. United States Patent US4228803, March 1980.

[237] R. Rojas. *Neural Networks: A Systematic Introduction.* Springer-Verlag Berlin Heidelberg, Germany, 1996.

[238] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Cornell Aeronautical Laboratory, Buffalo, NY, 1961.

[239] Z. Ruan, J. Han, and Y. Han. Bp neural network implementation on real-time reconfigurable FPGA system for a soft-sensing process. In *Proceedings of the International Conference on Neural Networks and Brain (ICNN&B)*, volume 2, pages 959–963, 2005.

[240] A. L. Rusiecki. Fault tolerant feedforward neural network with median neuron input function. *Electronics Letters*, 41(10):603–605, 2005.

[241] V. Saichand, N. Devi.M, Arumugam. S, and N. Mohankumar. FPGA realization of activation function for artificial neural networks. In *Proceedings of the Eighth Interna-*

tional Conference on Intelligent Systems Design and Applications (ISDA), volume 3, pages 159–164, 2008.

[242] S. M. Saif, H. M. Abbas, and S. M. Nassar. An FPGA implementation of a competitive Hopfield neural network for use in histogram equalization. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 2815–2822, 2006.

[243] A. P. Salvatore, N. A. Thorne, C. M. Gross, and M. P. Cannito. Neural network approach to speech pathology. In *Proceedings of the 42nd Midwest Symposium on Circuits and Systems*, volume 1, pages 439–442, 1999.

[244] P. K. Samudrala, J. Ramos, and S. Katkoori. Selective triple modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs. *IEEE Transactions on Nuclear Science*, 51(5):2957–2969, 2004.

[245] G. Santhanam, S. I. Ryu, B. M. Yu, A. Afshar, and K. V. Shenoy. A high-performance brain-computer interface. *Nature*, 442(7099):195–198, 2006.

[246] S. Satyanarayana. *Analog VLSI Implementation of Reconfigurable Neural Networks.* PhD thesis, Columbia University, New York, NY, 1991.

[247] S. Satyanarayana, Y. P. Tsividis, and H. P. Craf. Analogue neural networks with distributed neurons. *Electronics Letter*, 25(5):302–304, 1989.

[248] S. Satyanarayana, Y. P. Tsividis, and H. P. Craf. A reconfigurable analog VLSI neural network chip. In *Advances in Neural Information Processing Systems 2*, volume 2, pages 758–768, 1990.

[249] S. Satyanarayana, Y. P. Tsividis, and H. P. Craf. A reconfigurable VLSI neural network. *IEEE Journal of Solid-State Circuits*, 27(1):67–81, 1992.

[250] A. W. Savich, M. Moussa, and S. Areibi. The impact of arithmetic representation on implementing MLP-BP on FPGAs: A study. *IEEE Transactions on Neural Networks*, 18(1):240–252, 2007.

[251] T. Schoenauer, A. Jahnke, U. Roth, and H. Klar. Digital neurohardware: Principles and perspectives. In *Proceedings of the Neuronal Networks in Applications*, pages 101–106, 1998.

[252] J. Schumann and Y. Liu. *Applications of Neural Networks in High Assurance Systems.* Springer-Verlag New York, LLC, New York, NY, 1st edition, 2010.

[253] C. H. Séquin and R. D. Clay. Fault tolerance in artificial neural networks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, volume 1, pages 703–708, 1990.

[254] C. H. Séquin and R. D. Clay. *Neural Networks: Concepts, Applications and Implementations*, volume 4, chapter Fault Tolerance in Feed-Forward Artificial Neural Networks, pages 111–141. Prentice-Hall, New Jersey, 1991.

[255] J. Shadbolt and J. G. Taylor. *Neural Networks and the Financial Markets: Predicting, Combining and Portfolio Optimisation*. Springer-Verlag New York, LLC, New York, NY, 1st edition, 2002.

[256] H. Shayani, P. J. Bentley, and A. M. Tyrrell. Hardware implementation of a bio-plausible neuron model for evolution and growth of spiking neural networks on FPGA. In *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 236–243, 2008.

[257] X. She and K. S. McElvain. Time multiplexed triple modular redundancy for single event upset mitigation. *IEEE Transactions on Nuclear Science*, 56(4):2443–2448, 2009.

[258] K. Shima and T. Tsuji. FPGA implementation of a probabilistic neural network using delta-sigma modulation for pattern discrimination of EMG signals. In *Proceedings of the IEEE/ICME International Conference on Complex Medical Engineering (CME)*, pages 402–407, 2007.

[259] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi. Modeling the effect of technology trends on the soft error rate of combinational logic. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, pages 389–398, 2002.

[260] D. P. Siewiorek and R. S. Swarz. *Reliable Computer Systems: Design and Evaluation*. A K Peters, Ltd., Natick, MA, 3rd edition, 1998.

[261] R. Silipo and C. Marchesi. Artificial neural networks for automatic ECG analysis. *IEEE Transactions on Signal Processing*, 46(5):1417–1425, 1998.

[262] G. J. Snoek, M. J. IJzerman, F. A. in't Groen, T. S. Stoffers, and G. Zilvold. Use of the NESS handmaster to restore handfunction in tetraplegia: clinical experiences in ten patients. *Spinal Cord*, 38(4):244–249, 2000.

[263] A. Soares, A. Andrade, E. Lamounier, and R. Carrijo. The development of a virtual myoelectric prosthesis controlled by an EMG pattern recognition system based on neural networks. *Journal of Intelligent Information Systems*, 21(2):127–141, 2003.

[264] D. F. Specht. Probabilistic neural networks. *Neural Networks*, 3(1):109–118, 1990.

[265] V. Srinivasan, C. Eswaran, and N. Sriraam. Approximate entropy-based epileptic EEG detection using artificial neural networks. *IEEE Transactions on Information Technology in Biomedicine*, 11(3):288–295, 2007.

[266] E. Sugawara, M. Fukushi, and S. Horiguchi. Fault tolerant multi-layer neural networks with GA training. In *Proceedings of the 18th International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 328–335, 2003.

[267] M. Sun and R. J. Sclabassi. The forward EEG solutions can be computed using artificial neural networks. *IEEE Transactions on Neural Networks*, 47(8):1044–1050, 2000.

[268] Y. Sun. RNA: Resuable neuron architecture for on-chip electrocardiogram classification and machine learning. Master's thesis, University of Pittsburgh, Pittsburgh, PA, 2009.

[269] J. A. K. Suykens, J. P. L. Vandewalle, and B. L. Moor. *Artificial Neural Networks for Modelling and Control of Non-Linear Systems.* Springer-Verlag New York, LLC, New York, NY, 2010.

[270] H. Takase, H. Kita, and T. Hayashi. Weight minimization approach for fault tolerant multi-layer neural networks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, volume 4, pages 2656–2660, 2001.

[271] H. Takase, T. Shinogi, T. Hayashi, and H. Kita. Evaluation function for fault tolerant multi-layer neural networks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, volume 3, pages 521–526, 2000.

[272] A. M. Tan, W. Zhang, and J. M. Levine. NG2: A component of the glial scar that inhibits axon growth. *Journal of Anatomy*, 207(6):717–725, 2005.

[273] Y. Taniguchi, N. Kamiura, Y. Hata, and N. Matsui. Activation function manipulation for fault tolerant feedforward neural networks. In *Proceedings of the Eighth Asian Test Symposium (ATS)*, pages 203–208, 1999.

[274] S. Tatikonda and P. Agarwal. Field programmable gate array (FPGA) based neural network implementation of motion control and fault diagnosis of induction motor drive. In *Proceedings of the International Conference on Industrial Technology (ICIT)*, pages 1–6, 2008.

[275] E. Taub, G. Uswatte, and T. Elbert. New treatments in neurorehabiliation founded on basic research. *Nature Reviews Neuroscience*, 3(3):228–236, 2002.

[276] D. M. Taylor, S. I. Helms Tillery, and A. B. Schwartz. Direct cortical control of 3d neuroprosthetic devices. *Science*, 296(5574):1829–1832, 2002.

[277] E. B. Tchernev, R. G. Mulvaney, and D. S. Phatak. Investigating the fault tolerance of neural networks. *Neural Computation*, 17(7):1646–1664, 2005.

[278] J. Teifel. Self-voting dual-modular-redundancy circuits for single-event-transient mitigation. *IEEE Transactions on Nuclear Science*, 55(6):3435–3439, 2008.

[279] Texas Instruments Incorporated. *TMS320C64x DSP Library Programmer's Reference.* SPRU565B, Dallas, TX, October 2003.

[280] T. Theocharides, G. Link, N. Vijaykrishnan, M. J. Irwin, and V. Srikantam. A generic reconfigurable neural network architecture as a network on chip. In *Proceedings of the International SOC Conference*, pages 191–194, 2004.

[281] A. Tisan, S. Oniga, B. Attila, and G. Ciprian. Architecture and algorithms for synthesizable neural networks with on-chip learning. In *Proceedings of the International Symposium on Signals, Circuits and Systems (ISSCS)*, volume 1, pages 1–4, 2007.

[282] K. Y. Tong and M. H. Granat. Gait control system for functional electrical stimulation using neural networks. *Medical and Biological Engineering and Computing*, 37(1):35–41, 1999.

[283] K. Y. Tong and M. H. Granat. Reliability of neural-network functional electrical stimulation gait-control system. *Medical and Biological Engineering and Computing*, 37(5):633–638, 1999b.

[284] A. Upegui. *Dynamically Reconfigurable Bio-Inspired Hardware*. PhD thesis, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, 2006.

[285] A. Upegui, C. A. Peña-Reyes, and E. Sanchez. A methodology for evolving spiking neural-network topologies on line using partial dyanmic reconfiguration. In *Proceedings of the International Conference on Computational Intelligence*, 2003.

[286] A. Upegui, C. A. Peña-Reyes, and E. Sanchez. An FPGA platform for on-line topology exploration of spiking neural networks. *Microprocessors and Microsystems*, 29(5):211–223, 2005.

[287] A. B. Usakli, S. Gurkan, F. Aloise, G. Vecchiato, and F. Babiloni. On the use of electrooculogram for efficient human computer interfaces. *Computational Intelligence and Neuroscience*, 2010:5 pages, 2010.

[288] E. A. Vittoz. Analog VLSI implementation of neural networks. In *Proceedings of IEEE Symposium on Circuits and Systems*, pages 2524–2527, 1990.

[289] J. von Neumann. *The Computer and The Brain*. Yale University Press, New Haven, CT, 1st edition, 1959.

[290] M. Wakamura and Y. Maeda. FPGA implementation of Hopfield neural network via simultaneous perturbation rule. In *Proceedings of the SICE Annual Conference*, volume 2, pages 1272–1275, 2003.

[291] N. Wanas, G. Auda, M. S. Kamel, and F. Karray. On the optimal number of hidden nodes in a neural network. In *Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, volume 2, pages 918–921, 1998.

[292] L. Wang and T. S. Buchanan. Predication of joint moments using a neural network model of muscle activations from EMG signals. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 10(1):30–37, 2002.

[293] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Generalization by weight-elimination with application to forecasting. In *Proceedings of the Conference on Advances in Neural Information Processing Systems 3*, pages 875–882, 1990.

[294] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Generalization by weight-elimination applied to currency exchange rate precition. In *Proceedings of the International Joint Conference on Neural Networks*, volume 1, pages 837–841, 1991.

[295] J. Wessberg, C. R. Stambaugh, J. D. Kralik, P. D. Beck, M. Lauback, J. K. Chapin, J. Kim, S. J. Biggs, M. A. Srinivasan, and M. A. L. Nicolelis. Real-time prediction of hand trajectory by ensembles of cortical neurons in primates. *Nature*, 408(6810):361–365, 2000.

[296] P. W. Wohlgemuth, D. Munneke, G. Stoop, H. Westendorp, and M. Rouw. System and method for classifying sensed atrial events in a cardiac pacing system. United States Patent US6556859, April 2003.

[297] W. P. Wohlgemuth. Cardiac pacing system with improved physiological event classification based on DSP. United States Patent US6029087, February 2000.

[298] D. F. Wolf, R. A. F. Romero, and E. Marques. Using embedded processors in hardware models of artificial neural networks. In *Proceedings of SBAI – Simpósio Brasileiro de Automao Inteligente*, pages 78–83, 2001.

[299] Y. Wu, Q. Song, X. Yang, and L. Lan. Recurrent neural network control of functional electrical stimulation systems. In *Proceedings of the International Conference on Biomedical and Pharmaceutical Engineering (ICBPE)*, pages 400–404, 2006.

[300] Xilinx, Inc. *XtremeDSP for Virtex-4 FPGAs User Guide*. San Jose, CA, UG073 (v2.7) edition, May 2008.

[301] Xilinx, Inc. *Virtex-5 Family Overview — Product Specification*. San Jose, CA, DS100 (v5.0) edition, February 2009.

[302] Xilinx, Inc. *Virtex-5 FPGA XtremeDSP Design Considerations User Guide*. San Jose, CA, UG193 (v3.3) edition, January 2009.

[303] L. Xu, S. Furukawa, and J. C. Middlebrooks. Auditory cortical responses in the cat to sounds that produce spatial illusions. *Nature*, 399(6737):688–691, 1999.

[304] T. Yamakawa, K. Horio, and T. Hiratsuka. Advanced self-organizing maps using binary weight vector and its digital hardware design. In *Proceedings of the 9th International Conference on Neural Information Processing (ICONIP)*, volume 3, pages 1330–1335, 2002.

[305] K. Yamamori, S. Horiguchi, J. H. Kim, S.-K. Park, and B. H. Ham. The efficient design of fault-tolerant artificial neural networks. In *Proceedings of the IEEE International Conference on Neural Networks (ICNN)*, volume 3, pages 1487–1491, 1995.

[306] S. X. Yang and M. Meng. An efficient neural network approach to dynamic robot motion planning. *Neural Networks*, 13(2):143–148, 2000.

[307] L. Yu, S. Wang, and K. K. Lai. *Foreign-Exchange-Rate Forecasting with Artificial Neural Networks*. Springer-Verlag New York, LLC, New York, NY, 2009.

[308] S. B. Yun, Y. J. Kim, S. S. Dong, and C. H. Lee. Hardware implementation of neural network with expansible and reconfigurable architecture. In *Proceedings of the 9th International Conference on Neural Information Processing (ICONIP)*, volume 2, pages 970–975, 2002.

[309] A. S. Zadgaonkar, A. Shukla, and A. Tiwari. Speech pathological study of epinosic patient using aritificial neural network. In *Proceedings of the First Regional Conference of the IEEE Engineering in Medicine and Biology Society*, pages 3/85–3/86, 1995.

[310] D. Zhang and H. Li. A stochastic-based FPGA controller for an induction motor drive with integrated neural network algorithms. *IEEE Transactions on Industrial Electronics*, 55(2):551–561, 2008.

[311] J. Zhu, G. J. Milne, and B. K. Gunther. Towards an FPGA based reconfigurable computing environment for neural network implementations. In *Proceedings of the 9th International Conference on Artificial Neural Networks (ICANN)*, volume 2, pages 661–666, 1999.

[312] J. Zhu and P. Sutton. FPGA implementations of neural networks – a survey of a decade of progress. In P. Y. K. Cheung, G. A. Constantinides, and J. T. de Sousa, editors, *Field-Programmable Logic and Applications*, volume 2778 of *Lecture Notes in Computer Science*, pages 1062–1066. Springer, Berlin/Heidelberg, Germany, 2003.