2022-12-01

Optimized Learning Using Fuzzy-Inference-Assisted Algorithms For Deep Learning

Miroslava Barua
University of Texas at El Paso

OPTIMIZED LEARNING USING FUZZY-INFERENCE-ASSISTED ALGORITHMS FOR DEEP LEARNING

MIROSLAVA BARUA OLIVO

Doctoral Program in Electrical and Computer Engineering

APPROVED:

Patricia A. Nava, Ph.D., Chair

Miguel Velez-Reyes, Ph.D.

Robert C. Roberts, Ph.D.

Fernando R. Jiménez Arévalo, Ph.D.

Stephen L. Crites, Jr., Ph.D.
Dean of the Graduate School

DEDICATION

To my parents, my family, my friends and my advisor
for their love, unconditional support, endless help,
infinite kindness, for being a constant source of inspiration
and serving as the guiding light in my life.

And to everyone that cherishes being a lifelong learner:
remember that everyone should have the opportunity
to transform their life through learning
… even machines.

Let’s keep on learning!

OPTIMIZED LEARNING USING FUZZY-INFERENCE-ASSISTED ALGORITHMS FOR DEEP LEARNING

by

MIROSLAVA BARUA OLIVO, MSEE, BSEE

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

Department of Electrical and Computer Engineering

THE UNIVERSITY OF TEXAS AT EL PASO

December 2022

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to everyone that has been part of this journey; you are too many to mention, but please know that because of your support, encouragement and your help we have together finally arrived to the completion of this dissertation!

I would like to thank my parents, with all my heart, for being an example of what love is capable of doing for others, for their constant support, and for helping me grow spiritually to firmly believe that we never walk alone because God is always with us. Thank you to my twin sister Maru, for helping me with absolutely anything I need, for always leading the way, for being the voice of reason to my crazy ideas, and for showing me that together we are ready to face anything that comes our way. Thank you to Aydé for joining me in this ride from the start, for constantly cheering for me, for always inspiring me with your kind and generous heart and encouraging to keep going. Thank you to Anel and Diana for always being there for me, for your unconditional support and for helping me believe we can chase our wildest dreams. Thank you to Virginia for sharing this journey with me and for all the encouragement.

My sincerest gratitude to my dissertation committee members Dr. Miguel Velez-Reyes, Dr. Robert Roberts and Dr. Fernando Jiménez Arévalo for your participation; your time; your advice; your kind words of encouragement; for being an inspiring example of dedication, hard work and service; and for genuinely supporting all students, including me. Thank you so much.

I am profoundly grateful to Dr. Patricia A. Nava for being my mentor, advisor, teacher, role model and friend. Thank you for fulfilling each role with the utmost dedication and for being an inspiration by how you carry yourself professionally and personally. You are an example I will always try to follow. Thank you for your endless support, for your constant help, for your encouragement and for always believing in me, even when I didn't. I will never find enough words to thank you for all that you do for me. It has been a privilege, a pleasure and an honor to work under your guidance.

Thank you to all, but most importantly, thank you to God for all these blessings.

ABSTRACT

For years, researchers in Artificial Intelligence (AI) and Deep Learning (DL) observed that performance of a Deep Learning Network (DLN) could be improved by using larger and larger datasets coupled with complex network architectures. Although these strategies yield remarkable results, they have limits, dictated by data quantity and quality, rising costs by the increased computational power, or, more frequently, by long training times on networks that are very large.

Training DLN requires laborious work involving multiple layers of densely connected neurons, updates to millions of network parameters, while potentially iterating thousands of times through millions of entries in a big dataset. Reducing DLN training time is an important challenge to address and it is the goal of this research. This study provides innovations at the learning algorithm level to improve the efficiency of the training process; specifically, it optimizes the Backpropagation (BP) algorithm by using fuzzy-inference assisted learning to reduce the number of required operations completed during the training phase while at the same time maintaining performance accuracy. The created two-phase fuzzy inference system is integrated into the BP algorithm to provide decision support, and when appropriate, utilize a speed-up technique of skipping training operations. The results for the proposed model trained with benchmark datasets show remarkable savings of up to 82%, effectively reducing the execution time and accomplishing the desired speedup, at times reaching convergence 600 epochs earlier than baseline case which provides considerable extra savings and optimizes training even further. Remarkably, FIL-BP model reaches same level of system error minimization as the traditional implementation; it achieves same high classification accuracy, and it improves generalization capability by averting unnecessary weight updates that result from overtraining. The speed-up of the training process provides savings that increase as the complexity, size, and challenge of the dataset increases.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ILLUSTRATIONS

CHAPTER 1: INTRODUCTION

1.1 PROBLEM STATEMENT

Exciting applications of Artificial Intelligence (AI) and Deep Learning (DL) that have achieved great results are usually implemented on high-performance Deep Learning Networks (DLN) that are enabled by scale, as well as the continuous improvements in the large datasets, the complex architectures, and the efficiency of powerful training algorithms.

In the last twenty years, digitization of human activity through the use of computers, mobile applications, inexpensive sensors, and other digital devices gave rise to the collection of large datasets typically referred to as big data. Researchers observed that better network performance could be achieved by leveraging the size of the datasets coupled with larger and larger networks, capable of handling the scale of the data with their increased computational power implemented in dense pipelined or parallel systems. Although these strategies yielded remarkable results, they have limits, dictated by data quantity and quality, or, more frequently, by long training times on networks that are very large.

Training a network requires laborious work, involving multiple layers of densely connected neurons, updates to numerous network parameters, while potentially iterating thousands of times through millions of entries in a big dataset. Some applications are not able to afford long training sessions which, depending on the problem and implementation, might take undesirable amounts of time to complete. Another reason for seeking faster training is to enhance the iterative AI innovation cycle and its productivity. New implementation ideas are constantly emerging and they need to be tested to evaluate their performance and, when necessary, go back and make modifications to the details and train again. But when the experiment results take too long to

obtain, this hinders productivity and it makes it less likely to discover if such idea fits an application.

Making progress in trying to reduce DLN training time is an important challenge to address and it is the goal of many studies, including this one. Researchers in academia, government, and industry along with big tech companies such as Google, IBM, Amazon, Nvidia, Intel, Qualcomm, Facebook, and Microsoft have invested a lot of resources in optimizing DLNs. Previous optimization studies and techniques have focused on network architectural implementations using GPUs, CPU accelerators and other expensive specialized hardware designed for training computations. However, optimization could also happen at the learning algorithm level, consequently working in algorithm innovations is a promising scheme to help networks train faster.

1.2 Research Purpose

The goal of this study is to modify the learning process used in DLNs by inserting an intelligent agent in order to speed up the network. Specifically, this research proposes to optimize the Backpropagation (BP) algorithm by using fuzzy-inference assisted learning to reduce the number of required operations completed during the training phase while at the same time maintaining performance accuracy.

The contributions of this work are the development of a framework to implement a feedforward multilayer network trained with both the traditional and modified BP models. For the modified model, a two-phase fuzzy inference system was created and integrated into the BP algorithm to provide decision support, and when appropriate, utilize a speed-up technique of skipping training operations.

The performance of the network under both training models will be evaluated by using three databases that are different in size and complexity. The resulting savings given by the amount of training operations skipped and the accuracy of the modified model will be compared to the traditional BP implementation in order to determine the benefits of the speed-up strategy.

1.3 Dissertation Outline

The next chapters present the research work in the following manner. Chapter 2 provides the necessary background information on AI and the different strategies that can be used to improve performance; an explanation of how neural networks were developed, how they work, and how they are designed; and a description of fuzzy logic, its components, and how to design a fuzzy inference system. Chapter 3 offers the literature review that explores the previous work in this research area to show the trends and results that have been used to optimize training of DLNs. Chapter 4 presents the implementation details including the design choices made for the baseline and modified models; an in-depth explanation of how the BP algorithm works; the strategy used to insert the fuzzy inference assistance into the traditional algorithm; and an overview of the integration of all of these components into program code. Chapter 5 provides the results obtained after training the network with three different databases under the two learning models and a comparison of their respective performance. Chapter 6 offers the conclusions reached and suggested recommendations for future work that can enhance performance and increase the benefits of using the proposed model

CHAPTER 2: BACKGROUND

2.1 Artificial Intelligence

Artificial Intelligence (AI) is the overarching term used to encompass the subfields of Machine Learning (ML), Neural Networks (NN) and Deep Learning (DL) as shown in Figure 2.1 below. Although people sometimes use these terms interchangeably, it is important to understand the differences between them.

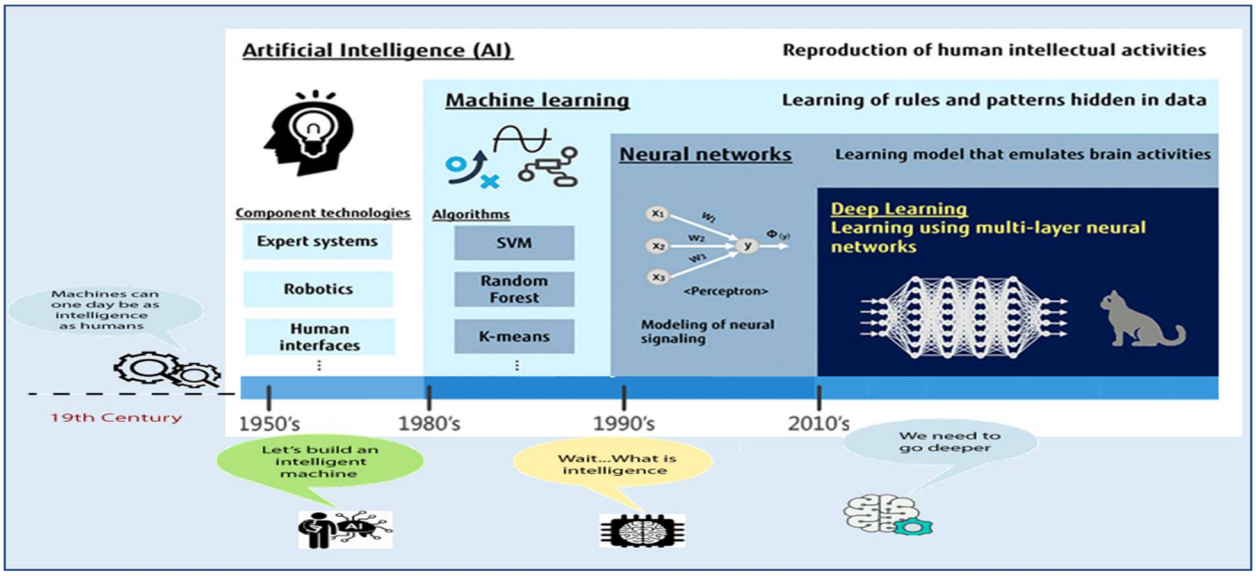


Figure 2.1: Evolution of Artificial Intelligence and Associated Subfields [Fal21]

AI is not a new term, it was first used in the field back in 1956 when a researcher named John McCarthy coined the term *Artificial Intelligence* to refer to the science and engineering of making computing systems that use datasets, iterative processing and intelligent algorithms to perform tasks usually reserved for human intelligence including reasoning, interacting, and making decisions.

ML uses more advanced algorithms to give computers the ability to learn from data, without being explicitly programmed to do so, and make informed decisions about what is learned.

ML can accomplish tasks such as playing chess or giving users a personalized recommendation in ecommerce sites or streaming services.

Artificial NN are ML systems inspired by biological neural networks (a.k.a. the brain). They use multiple computational models called *neuron*s which are connected in layers to learn the relationships between inputs and outputs and extract abstract features from the data in a hierarchical fashion. The next major section of this chapter describes how NNs work.

DL use very large NNs with multiple layers to create what is called Deep Neural Networks (DNN) or Deep Learning Networks (DLN). The word *deep* refers to NN structures (with three or more *hidden* layers as opposed to single hidden layers of shallow networks) that can learn from vast amounts of data to make intelligent decisions on its own. DL is the most human-like artificial intelligence that we have today. The attractiveness of DL is the ability to automatically learn or extract the features from datasets without using traditional methods.

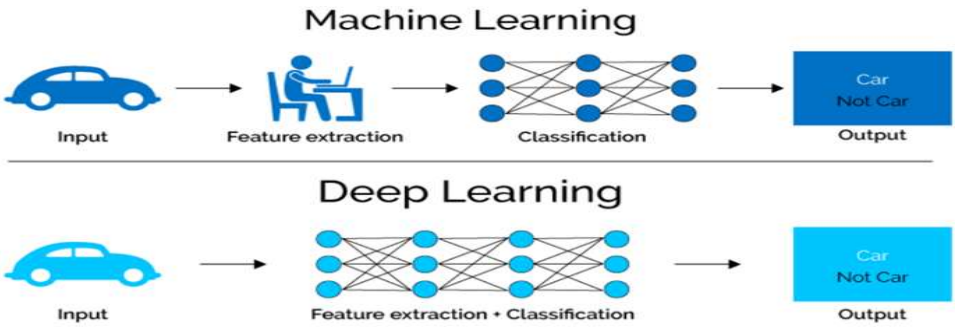


Illustration 2.1: How ML and DL achieve feature extraction [Kau20]

In essence, to build an AI system we need three items: code + hardware + data

- the code implements the architecture of the model and the *learning algorithm*
- the *hardware* provides the platform for the code to run and perform all the required computations

- the *data* is used by the system to extract information and learn.

When a system is not performing to a desired standard, AI researchers focus on making improvements to one of those three items, as discussed in detail in the following sections.

2.1.1 Hardware changes to improve AI performance

Some researchers and developers make changes to the AI system implementation hardware by either introducing more units to increase the computational power and benefit from parallelization of tasks or by using hardware accelerators to handle the size of the model/data more efficiently. Traditionally CPUs are used for small models, small datasets or in applications that require the flexibility of dynamic programming such as those in space exploration. Depending on the application implementation, some researchers rely on the advantage of replacing CPUs with chips more suitable for parallelization tasks such as FPGAs and GPUs, and for larger implementations it is typical to use AI specific platforms such as TPUs and cloud services.

FPGAs allow algorithms to be downloaded into the hardware multiple times so they provide reconfigurable flexibility at low power consumption. They are typically used for large datasets and large models in applications that require intensive computations and efficiency in performance-per-watt. Some companies such as Microsoft use FPGAs to accelerate AI processes for its cloud service Azure and it has been predicted that in the next year about a third of the AI cloud service providers will use FPGAs to accelerate its nodes.

GPUs were originally created for quick response in image and video processing of large amounts of data typically found in video games, but they have been extensively and successfully used in AI applications that have medium to large models and/or datasets. Due to higher demand

and suitability for AI, companies such as Nvidia, Qualcomm and AMD have made many improvements on GPU performance and reduced the cost of each unit.

Larger applications benefit from the use of highly efficient ASICS specifically made to accelerate AI processes such as the Tensor Processing Unit (TPUs) which was developed by Google and tailored to use TensorFlow. TPUs have high throughput of low precision arithmetic and are usually used in clusters of thousands of units available via AI cloud services.

Cloud services are for quick model training and deployment of high accuracy systems that require training from datasets with millions of labeled examples and when cost of using the services doesn't become an issue for the user. Systems such as Amazon's Machine Learning Services, Microsoft's Azure Machine Learning, Google Cloud and IBM's Watson's Machine Learning rely on supercomputers with thousands of units that provide high computing power in nods or pods. Cloud services allows researchers and developers to rent "slices" of such machines at variable costs according to the resources used and the time consumed. For instance, a standard ResNet-50 image classification model that has over 50 layers and over 23 million trainable parameters using the ImageNet dataset can be trained by a single Google TPU pod in around 302 minutes, but it takes 11.3 minutes in a version 2 pod and only 7.1 minutes in a v3 pod [Lar19].

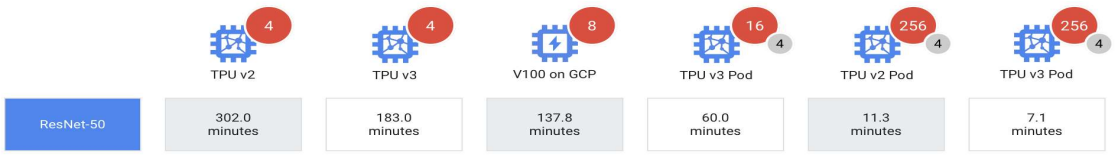


Illustration 2.2: Training time of ResNet-50 on Google's Cloud TPU pods [Lar19].

The cost to train an AI system is on the rise. For instance, a system that requires ten days of training on Google Cloud platform at a cost of $2.28 per hour will cost over $300,000. This cost can become stratospheric even for Google's subsidiary DeepMind that reportedly spent $35

million to train an AI system to learn to play the game of AlphaGo, but when the same company wanted to train a model to play StarCraft, they had to abandon their efforts because training cost would have been too high to be feasible [Wig21].

The cost of developing, implementing and maintaining a custom AI system in a cloud service varies on different factors such as platform, size and quality of data combined with complexity of the system and the required accuracy. Training state-of-the-art AI systems have increasingly large costs that are feasible only to a few large companies and government agencies and can leave behind the startup companies, academics and students that might not be able to afford such services. For instance, training an AI language model called Megatron Turing Natural Language Generation with 530 billion parameters to provide reading comprehension and natural language interfaces came at a cost of millions of dollars [Wig21]. A general video/speech analysis for telemedicine can cost around $36-56 thousand, intelligent recommendation engine between $20 – 35 thousand [Klu22]. By 2030, AI could contribute up to $15.7 trillion to the global economy.

AI Cloud services also come with environmental costs because training systems consumes increasingly large amounts of energy. For example, when Google used thousands of core processors to run a system that was capable of detecting cats on YouTube videos, these large amounts of processors consumed so much energy that it needed to be liquid-cooled. The University of Massachusetts reported, in a 2019 paper [Ho19], that the training effort for a large language model emitted five times more carbon dioxide than an average vehicle over its entire lifetime.

2.1.2 Making improvements on the data

Sometimes researchers try to improve AI system performance by focusing on how to systematically change the data by either collecting larger amounts of examples, or doing data preparation by cleaning/pre-processing the data (on the inputs or labels) so it results in a higher quality data set. In fact, much of the progress in the last decades has been driven by using very large benchmark datasets.

Dr. Fei Fei Li is a pioneer and one of the world's leading experts on computer vision who is also a leading voice regarding ethical issues in AI. In 2003, she started her research at Princeton and continued at Stanford. Dr. Li had the intuition that, in order to help AI systems be able to "see," these systems first needed to have a very large (very *deep*) and high-quality database for training. By 2009 she had built two seminal databases that unleashed the power of large data: Caltech 101 and ImageNet [Lab21].

For Dr. Li, the idea of increasing the amount of data being provided to train AI systems was analogous to the way living organisms learn how to make sense of their environment. In other words, nobody tells an individual how to see, they are, rather, exposed to many images in the real world captured by their eyes and these serve as examples. Just like hearing is not the same as listening, taking many pictures does not help the individual to see. Seeing *begins* in the eyes but making sense of what its seen occurs in the brain. It took many years and more than 50,000 workers around 167 countries to collect, sort, and clean the nearly one billion candidate images to give life to ImageNet's impressive 15 million images, each one labeled in English and categorized into 22,000 classes. The full database was completely open and it is still available to the worldwide research community for free. ImageNet became the gold standard for machines to recognize images and be able to categorize different objects.

In addition to the database, an annual ImageNet Large Scale Visual Recognition Challenge was opened which has since consistently gathered the most state-of-the-art AI computer vision system solutions where teams showcase all their innovations. It was in 2018 when Alex Krizhevsky and Geoffrey Hinton won the competition by using a Convolutional Neural Network (CNN) architecture implemented on powerful GPUs and paired with this massive database to achieve a margin of error ten percentage points lower than any previous system [Kri17]. Their system is known as AlexNet and it became one of the most influential implementations in computer vision, in fact, many mark this milestone event as the start of the deep learning revolution.

Countless research teams followed the same recipe and obtained impressive results by using huge amounts of data combined with increasingly larger networks implementing the winning architecture. Very often, developers would improve the performance of a system by just adding larger amounts of training data or increasing the computational power. It is not unusual to hear that researchers spend more time collecting data than working on their models. It is not surprising to realize that continuing this strategy can only work up to a point, but eventually the system will be faced with physical and feasibility limits: you either run out of data or the added elements to the network will increase the number of parameters, becoming so large that it will take too long to train. It is not possible to continue to increase the computing power and the database size at the same rate as before, undoubtedly there will be a slowdown in advances and innovation. Therefore, now is the time for researchers to focus on improving the algorithms used by these solutions.

2.1.3 Learning algorithm changes to improve AI performance

Another important strategy to improve performance of AI systems is to hold the data and the hardware configuration fixed and make efforts to improve the learning process by making changes on the learning algorithm itself. The algorithm that has been most successful in DL applications is called Backpropagation (BP) algorithm. As mentioned in the previous section, CNNs have become the predominant type of neural network for image recognition and classification. These networks are trained using unsupervised learning with BP algorithm. The following chapter presents the literature review which goes over the trends and strategies that have been used to optimize NN training process. The implementation chapter goes over how the learning algorithms work, specifically BP. Please refer to both of those chapters for a detailed explanation of how modifications to the learning algorithm have improved the performance of AI systems.

As one can imagine, AI has progressed and evolved through the years enjoying periods of high activity (referred as *revolutions*) and survived years of lost confidence (called *winters*).

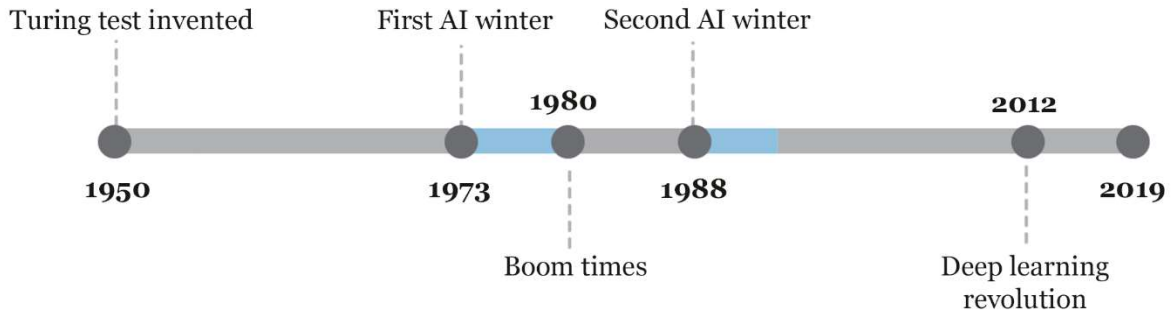


Illustration 2.3: AI progression through the years [Sch19]

Due to the high demand of DL solutions in areas such as edge computing, the Internet of Things (IoT) and the continuous growth in research studies, one can certainly predict that we are not on the verge of another winter period. In fact, almost daily news reports arise on the next great

AI system application, and the results big AI cloud services provides in diverse areas such as medicine, military, government, transportation, commerce, manufacturing, travel and entertainment, just to name a few.

The AI field continues to grow towards the goals of taking the systems to the next level. Current thought is that there are three stages of AI that can be envisioned for the current and future capabilities of AI. Our current AI systems are referred as having Artificial Narrow Intelligence (ANI) because they are specialized in one area and are designed to perform a single task. In the next stage of intelligence, we will have Artificial General Intelligence (AGI), which refers to computers that are as smart as humans across many tasks. The following stage is labeled as Artificial Super Intelligence (ASI) in which machines can experience consciousness and their intellect is much smarter than humans in practically every field.

AI innovation is also pushing boundaries to provide small-scale local solutions to applications that have little or no connection to the internet in devices such as cameras that can identify a person entering an area; drones and robots that need to react to what they face in their environment; on-board systems that allow autonomous vehicles to make quick decisions; and on-device portable solutions that provide secure facial or object recognition without data having to leave the device. Doing on-device training provides plenty of advantages including generalization opportunities and data privacy protection. This point is illustrated, for instance, by thinking of how to improve a vehicle's self-driving system. Just as people can enhance their development through continuous education, an AI system could greatly benefit, in the same way, from continuing to train by using real-world driver behavior data from the environment. If we think about collaborative learning (analogous to study groups), a centralized self-driving model could be trained on decentralized data by distributing the learning task to many independent vehicles. In

other words, by deploying the training model into many vehicles, each device can use its own environment real-world data to learn from, and after going through several thousand of examples and iterations, can send the results back so they can be combined to create a better model. When the updated model is ready, it can be deployed again to other several vehicles to test it with new independent data and continue the training cycle. This collaborative learning strategy could be used not just for vehicles, but on any other implementation that would benefit from learning from multiple sources and at the same time keeping the data locally to maintain privacy. This could be achieved by deploying it via smartphones, tablets, smart wearables, or recruiting multiple entities for hard-to-collect data such as hospitals, earthquake monitoring centers, power plants or even space-related missions.

Keeping the training data locally on the device helps maintain data privacy and allows real-time processing so learning actually becomes easier if it is kept in the same place where training is actually occurring, rather than sending data back to a server (think about Alexa, Siri and other personal assistants that send data back and forth, which can disrupt user experience). In order for this to occur, the training and validation steps need to happen on the individual devices; this can be enabled by optimizing the training algorithms.

In closing, the common goal in AI is to take part in human-like decision-making roles within a wide range of activities. The purpose of creating these intelligent systems *is not* to replace humans, but rather to assist, enhance, augment, and provide insights to their human counterparts. For example, intelligent systems are being used in the field of medicine to help doctors and nurses by providing extra pairs of tireless eyes to help them diagnose and take care of patients. A machine can be used to go over thousands or millions of examples and be able to extract information to make a decision. This approach would be very tiresome or overwhelming for the human

counterpart who might not be able to obtain a clear perception of the crucial information contained in those images. The expectation is that we will not only use these machines for their intelligence, but that we will also be able to collaborate with them in ways we perhaps haven't even imagined.

2.2 Neural Networks

The goal of neural networks is to achieve artificial intelligence by creating an information-processing system inspired by the way the human brain works.

From neuroscience we know that the structure of the brain involves many neurons interconnected in a loose, flexible, overlapping manner that allow the parallel processing needed for multitasking. The brain is made of approximately 100-billion neurons that have the ability to gather and transmit electrochemical signals (or brain waves) from one to another over long distances, communicating information about emotions and everything that it's seen, heard, tasted, touched, and smelled. Each biological neuron has three basic parts [Fre01]: the body of the neuron is called soma and it is where all the inputs are added or combined and when the summation of this signals is strong enough, the neuron will *fire* a response that is sent via the axon. The axon is a long projection that provides the transportation of the electrochemical signal towards neighboring neurons which are interconnected with each other neurons via nerve endings called dendrites. Neurons get connected where an axon meets a dendrite, and there exists a gap called synaptic gap or *synapse* that converts the activity from the axon into electrochemical effects that inhibit or excite activity in the connected neurons. If a neuron receives or 'sums' enough excitatory input signals that together are larger than the inhibitory input, the neuron will send a spike of

electrical activity of its own. This chain reaction is potentially transmitted through millions of cells.

It is important to understand that in the brain, learning occurs by changing the effectiveness of the synapses so that the influence of one neuron on another changes accordingly. With increased brain activity, certain patterns of connection are strengthened, making each connection easier to create next time [Chu01]; this is how memory develops. Memory is not just the number of neurons used, but also the number of connections between all neurons.

Neuroscientists, psychologists, and even philosophers believe that the organization and connectivity of the brain holds all of the knowledge as a function of the connections existing among the neurons and the strength of those connections. The structure of the interconnection of neurons is flexible and allows continuous modification during the brain's lifetime. This ability to adapt to new information allows the brain to function even when faced with new unpredictable environments.

The brain structure is very complex because each of its hundred billion neurons can connect with as many as ten thousand neighbors creating about 1,000 trillion synaptic connections which are very hard to replicate. The complex wiring system of the brain easily surpasses the complexity of even the most advanced supercomputers. Artificial networks use the brain as a guideline taking key points and looking at the functionality to produce models that try to resemble their basic operation.

An artificial neural network is a mathematical model composed of a large number of processing elements called neurons. These neurons are highly interconnected to create a parallel distributed information processing structure. The model of the artificial neuron and its connections has the following features [Fau94]:

- The information processing elements (neuron) receive many signals and sums all the weighted inputs.
- Signals may be modified at the receiving synapse by a weight. This modification typically multiplies the signal being transmitted.
- Sufficient input causes the neuron to transmit a single output via links.
- Information processing is local and memory is distributed (Long-term memory resides in neuron's synapses or weights; short term memory corresponds to the signals sent by the neurons).
- A synapse's strength (weight value) may be modified by experience.
- Fault tolerance and ability to retrain in case of small damage.

The following Figure depicts a biological neuron on the left and the corresponding model for an artificial neuron on the right. The similarities between them are very straightforward to appreciate.

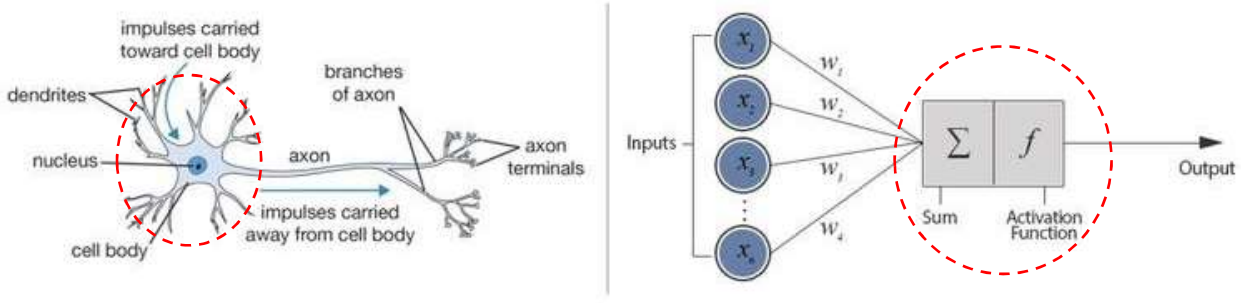


Figure 2.2: A biological neuron vs the mathematical model of an artificial neuron [Wil19].

In NN, each neuron is connected to other neurons via unidirectional signal channels called links, each with an associated weight ($W_{link\text{-}i}$) representing the information being used to solve the

problem. Inside the model of the artificial neuron there is an internal state or activation function, where the inputs received are processed. The neuron's function takes the weighted sum of the input signals and applies an operation to this value (function is usually nonlinear) which yields an output and "fires" only if the threshold level is met or passed.

The first artificial neuron model was designed in 1943, by neurophysiologist Warren McCulloch and logician Walter Pitts [Mcc43]. They connected many neurons into what is referred as a neural network. The neurons were developed as binary devices: output equal to 1 signifies that the neuron fired, an output equal to 0 means the neuron did not fire. This is summarized by the following equation:

$$y = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} w_i x_i \geq \theta \quad (2.1) \\ 0, & \text{if } \sum_{i=1}^{n} w_i x_i < \theta \quad (2.2) \end{cases}$$

where:

y is the output value of the neuron,
w is the value of the connecting weight,
x is the input value, and
θ is the threshold value.

The McCulloch-Pitts model continued to evolve and some of its shortcomings were addressed in updated models. For example, the problem with the McCulloch-Pitts model is that it provides no adaptation, and therefore has no ability to learn. In 1958, Frank Rosenblatt presented a solution to this problem with the adaptive perceptron that fires based on a binary step function with a fixed threshold.

To create a NN, a large number of neurons need to be connected together to form a working system. Designers must determine three important characteristics for their network: the type of *architecture* they want for their connections, the chosen *learning or training algorithm* and the *activation function* that the neuron will use to produce a response. Each of these characteristics are discussed next.

2.2.1 Architectures

The arrangement of the connections is referred as the architecture of the network and it can take different forms based on their corresponding purpose. There are too many architecture types to cover them all, but the most commonly used in NN arranges the neurons into multiple layers starting with all the input neurons lined up in an input layer, followed by the hidden layer(s) and finishing with an output layer. The Figure below shows a simple or shallow network with a single hidden layer and a DLNN showing its depth with four hidden layers.

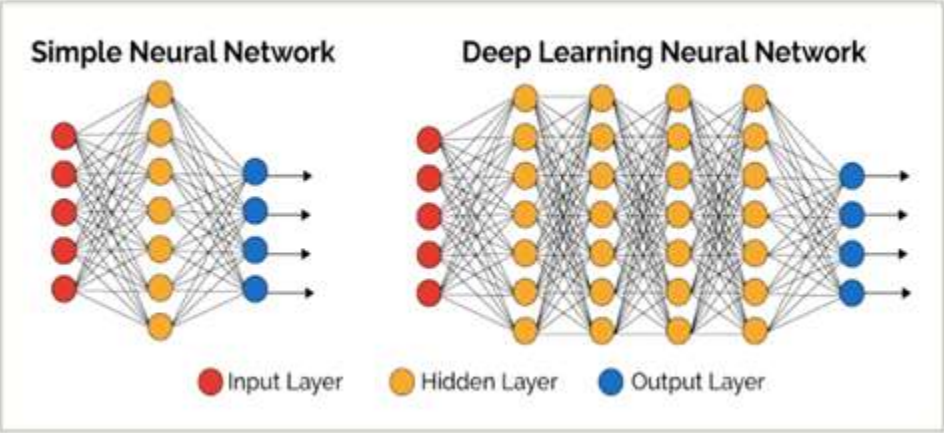


Figure 2.3: Neurons arranged into multilayers to form a simple NN and a DNN [Tch17].

Multilayer topologies are complex enough to be able to solve more difficult problems that a single-layer network can't solve. Typically having more hidden layers also means having more

computational power to complete laborious tasks required for training and evaluating the full network.

After arranging the neurons into layers, the interconnection pattern between the neurons will determine the type of architecture for the network.

If neurons are connected in a way that allows signals to flow from the input layer towards the output layer (from left to right) it is called a feed-forward network. If every neuron receives a connection link from every other neuron in the previous layer, the network is fully interconnected. If there are closed loops between neurons, either with itself or with a previous layer's neuron, it becomes a feedback or recurrent network. If they have convolutional and pooling layers, they are called convolutional NN. All the architectures mentioned are shown in the Figure 2.4 below.

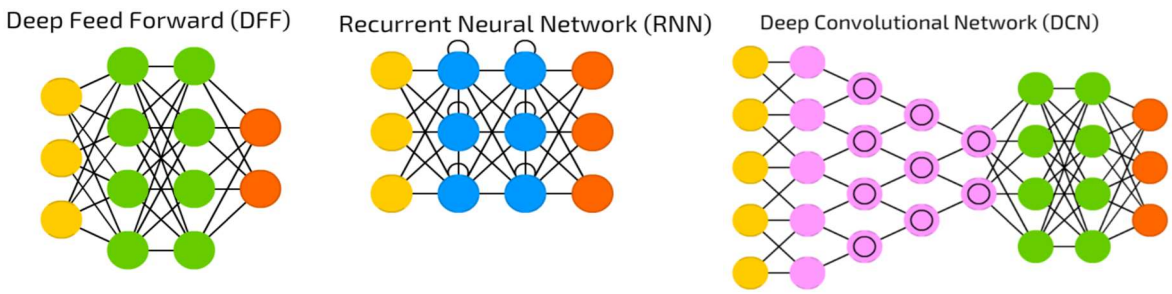


Figure 2.4: Different types of NN architectures [Tch17].

Feed-forward nets associate inputs with outputs; hence they are extensively used in pattern recognition. Feedback nets are very powerful for memory applications, but get extremely complicated quickly, due to the numerous interconnections. CNNs are used for image classification, image processing, video analysis and natural language processing tasks.

In a feed-forward multi-layer neural network, the connecting set of weights between the input and hidden layers determine when each hidden neuron is active. Therefore, the modification of these weights changes how the hidden neuron represents the input being received.

2.2.2 Training or Learning Algorithm

Once the network architecture is established, the next step is to determine the learning rule. Humans can learn in different ways by using exploration, sensation, moving from one experience to another, making mistakes and relying on repetition until an objective is achieved, finding patterns and making associations, or learning based on examples together with feedback from a teacher. Just like humans, DNNs use different training algorithms or learning rules to achieve their knowledge. Please note that the terms training and learning refer to the same process and are used interchangeably. DNN systems usually go through big datasets composed of thousands or millions of data samples divided into training and testing portions.

NNs have the ability of processing inputs to obtain an output; if the given response does not yield the desired result, the neuron's weights can be corrected during the learning or training process. This correction generally occurs by sequentially applying input values to the NN while making weight adjustments –according to the expected output– until the network converges or a minimal error is achieved. Modification of these weights as a function of 'experience' implies the use of a learning (training) rule. The knowledge obtained is stored in a weight matrix.

The type of training that a network uses depends on the type of problem to be solved. The training or learning algorithm provides a series of steps to be followed in a particular order. There are three major approaches to training or learning are [Jan97]:

- *Supervised learning*: This type of learning is characterized by datasets that have a known target output. A sequence of the training patterns is passed through the network and the expected output is also given. The external "teacher" checks the system's

response and compares it against the expected target. If the responses don't match, the weights are modified accordingly. In this manner, the net is trained to respond correctly. The goal is to obtain a set of weights that minimize the error. Learning occurs by adjusting the weights based on the training algorithm and following the learning rule. These weights are the coefficients of the hyperplane that is the decision surface that is used to distinguish between one class from another. Paradigms of supervised learning include Hebbian learning, delta rule and backpropagation algorithm and are usually applied to classification problems.

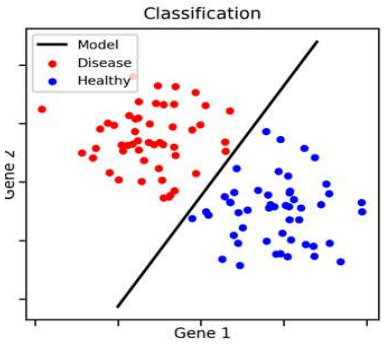


Illustration 2.4: Example of a classification problem [Tch17].

- *Unsupervised learning*: It is characterized by using dataset that only have inputs but no target output is provided, hence there is no "teacher" involved; the network iterates until it has its own output representation. This is also known as self-organization. The network is not given any knowledge about the expected output and it is trained to discover structures in the presented inputs. Paradigms of unsupervised learning include Kohonen self-organizing maps and competitive learning and they are usually applied to clustering problems.

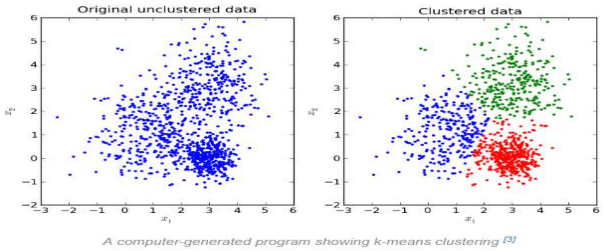


Illustration 2.5: Example of a clustering problem [Tch17].

- *Reinforcement*: The network is not provided with explicit output; instead, it is periodically given performance indicators, therefore, training uses trial and error. Neurons are given data samples and produce a solution and respond to feedback. If the network isn't responding as desired, the weights are changed by a random amount. No explicit teacher exists; the learning comes from lesson failures.

2.2.3 Activation Function

The response of a NN depends on the weights and activation function of each neuron. The basic operation of a neuron takes the weighted sum of the inputs and processes this number with the activation function to determine if the neuron will fire or not. There are three basic types of activation functions [Fau94]. The *identity or linear function*, where the output activity is proportional to the total weighted output. This can be expressed by the following equation:

$$f(x) = x, \text{ for all } x. \tag{2.3}$$

The second type is the *threshold or binary step* function, where the output is set at one of two levels, depending on whether the total input is greater than or less than some threshold value:

$$f(x) = \begin{cases} 1, & \text{if } x \geq \theta \quad (2.4) \\ 0, & \text{if } x < \theta \quad (2.5) \end{cases}$$

The third type is *sigmoid* (S-shaped curves) function, where the output varies continuously as the input changes. The binary sigmoid function is especially useful in backpropagation networks because the simple relationship between the value of the function at a point and the value of the derivative at that point reduces the computational burden during training. This function and its derivative are expressed by the following equations:

$$f(x) = \frac{1}{1 + \exp(-\sigma\ x)} \quad (2.6)$$

$$f'(x) = \sigma\ f(x)\ [1 - f(x)] \quad (2.7)$$

Unlike the multitasking brain, NNs are trained for a specific application and are able to "learn" and generalize from data, like humans do from experience. In theory, a neural network is capable of learning any mathematical function given a training dataset that is sufficient in size and quality.

Although there are many learning algorithms and diverse architectures, in our implementation we will investigate the backpropagation algorithm in great detail and will use a feed-forward multi-layer NN for our architecture. Backpropagation is still the most commonly used algorithm for learning and our chosen architecture is used extensively in DNN systems.

2.3 Fuzzy Logic

Lofti Zadeh is a professor in electrical engineering and computer science who felt that classical two-valued logic (yes/no) was too precise for many complex real-world problems and did not capture the different range of possibilities between *yes* and *no* that humans use for decision making. Hence, he generalized classic logic theory to develop Fuzzy Logic (FL), with the purpose of dealing with uncertainty. The FL method achieves machine intelligence by offering a method of representation and reasoning for complex human knowledge that is imprecise by nature. The following figure depicts the taxonomy of intelligent systems and the relations between AI, ANN and FL related to intelligent technology.

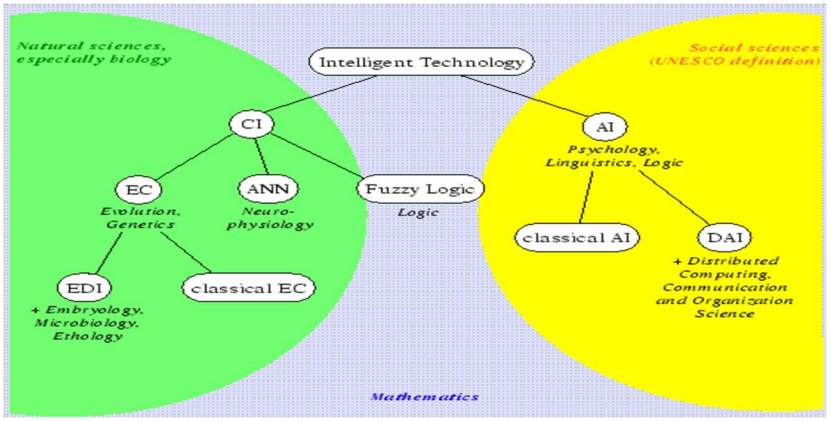


Figure 2.5: Taxonomy of intelligent systems [Sch02]

FL is a technology that enables designers to incorporate and implement intelligent control strategies based on human knowledge. Instead of relying on sharp boundaries, the control is based in fuzzy rules or heuristics. Some argue that FL is a clever disguise of probability theory, but

probability measures the likelihood of an event before the actual outcome is known while FL measures the degree to which an outcome belongs to a category that doesn't have a well-defined sharp boundary. For example, FL control is used in video recordings that use fuzzy logic-based image stabilization to determine if a particular image is a function of movement in the camera or is a result of movement of objects in the field of view.

There is a wide range of areas where successful fuzzy rule-based inference systems have been applied such as in control systems, system modeling and in industrial applications like consumer products, robotics, manufacturing, process control, medical imaging and financial trading to name a few.

2.3.1 Components of a FL inference system

FL is based on four important components: fuzzy sets, linguistic variables, possibility distribution, and fuzzy rules.

Unlike crisp classical sets where an element is either in the set or not (represented by value 1 or 0 respectively), *fuzzy sets* have smooth boundaries defined by *membership functions (μ)* that take values in the interval [0,1]. In other words, classic theory relies on sharp boundaries that require a particular threshold but in a fuzzy set membership is a matter of degree which allows for partial membership.

A membership function for a fuzzy set "A" on the universe U is defined by:

$$\mu_A : U \rightarrow [0,1] \tag{2.8}$$

This quantifies the degree of membership of the element in U to the fuzzy set A. A membership function should provide a gradual transition from regions completely outside the set to regions completely in the set. The degree of membership in a set is expressed by a value between zero

and one, where zero means it is entirely not in the set and 1 means it is completely in the set. Any number between zero and one award a partial degree of membership in the set.

Membership functions are graphs or curves that provide a gradual transition from regions completely outside a set to regions completely in the set, hence allowing a partial degree of membership and providing a convenient way to map input space to output space. There are different types, distinguished by the shape of the curve (triangular, trapezoidal, gaussian, bell-shaped and sigmoidal). The x-axis of the curve represents the universe and the y-axis represents the degree of membership.

Fuzzy sets are also associated with linguistically meaningful terms (such as cool, warm, hot, tall, short, fast, slow, etc.) that make it easier for human experts to express their knowledge using linguistic variables (such as *temperature, size, speed)*. Assigning a fuzzy set to a linguistic variable constrains the value of the variable to a matter of degree called possibility. A value can be described qualitatively by a linguistic term and quantitatively by the corresponding membership function. Because the meaning of a term depends on context, fuzzy sets are always defined in context to avoid misunderstandings.

The following figures show the graphical representation for a temperature controller example that uses Temperature as the input and Fan Speed as the output. Figure 2.6 shows the set of triangular membership functions described by linguistic terms *COLD*, *WARM* and *HOT*.

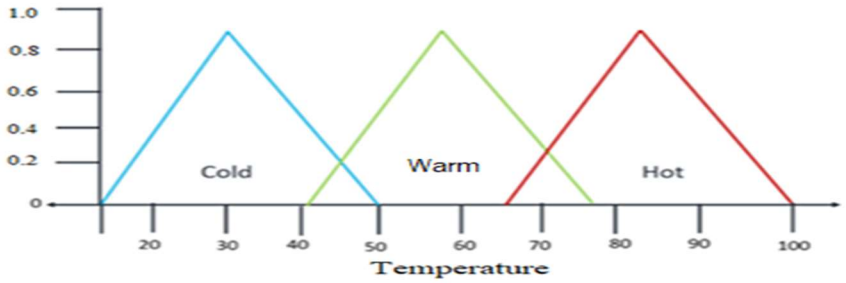


Figure 2.6: Membership functions for Temperature

Figure 2.7 corresponds to the output where it shows the fuzzy sets for the membership functions denoted by terms *SLOW, MEDIUM and FAST.*

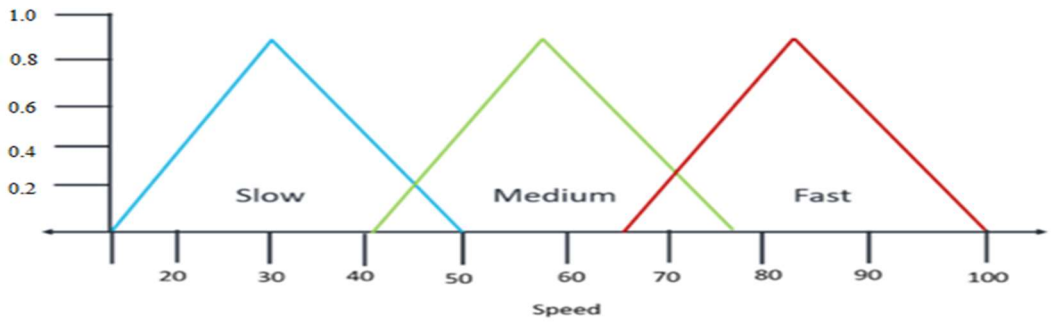


Figure 2.7: Membership functions for Speed

In FL, the basic unit for capturing knowledge is represented by *fuzzy if-then rules* (or fuzzy rules for short). A fuzzy rule has two components called *antecedent* and *consequent.* The antecedent is the *if* -part of the rule that describes an elastic condition that can be satisfied to a degree. The consequent is the *then*-part of the rule which is the conclusion that can be drawn when the condition holds. In other words, the output or the action that results from an antecedent being true corresponds to a strength reflecting the degree to which the antecedent is true.

In the previous example we could create a set of rules to determine the behavior of a controller such as:

If HOT *then* FAST
If WARM *then* MEDIUM
If COLD *then* SLOW

A fuzzy based system is constructed so that the generated output changes in a continuous manner regardless if the input crosses set borders.

2.3.2 Designing a Fuzzy System

Designing a fuzzy rule-based inference system requires the completion of three major steps to obtain a conclusion and make a decision.

The first step to complete is fuzzy matching or fuzzification of the input to calculate the degree to which input data matches the conditions of the fuzzy rules. This requires the calculation of the degree of membership by using functions to map each system input into one or more degrees of membership.

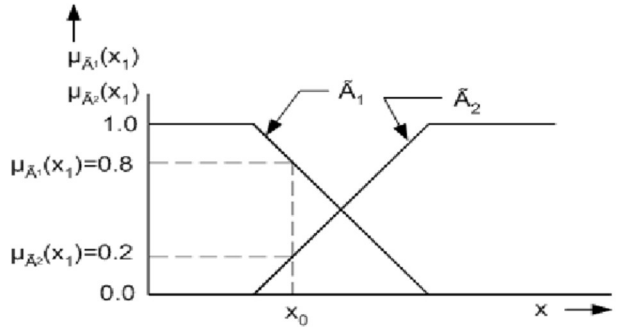


Illustration 2.6: Fuzzification of input values into degrees of membership

Typically, membership functions are triangular or trapezoidal because they are effective and efficient to use.

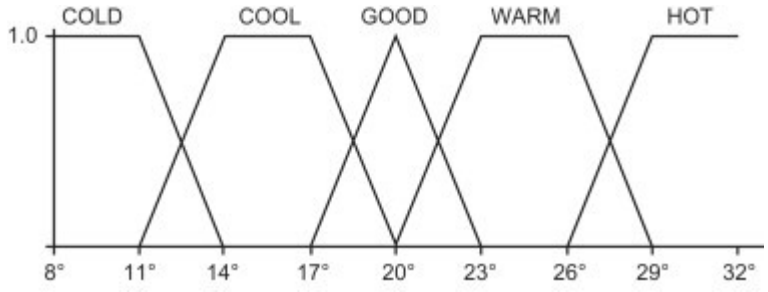


Illustration 2.7: Different fuzzy membership functions

The fuzzy sets must expand to cover all the x-axis and the rule of thumb is that there should be an overlap of about 25% between sets.

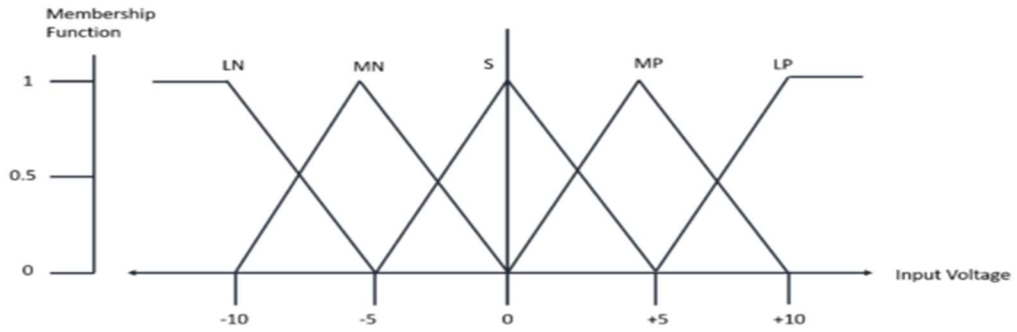


Illustration 2.8: Overlap between fuzzy sets

The second step is to complete the fuzzy rule-based inference or rule evaluation. In other words, the degrees of membership calculated in the previous step correspond directly to the "if" side of a rule containing the conditions (antecedent) and the "then" side that contains the consequence. To produce a conclusion two methods could be used: clipping method cuts off the top of the membership functions whose value is higher than the matching degree, or use the scaling method that scales down the membership function in proportion to the matching degree.

Very often when an input to the system triggers multiple fuzzy rules so the results need to be combined. When a rule has one or more antecedents, each will have a degree-of-truth or membership value assigned according to the fuzzification results. When this happens, rule evaluation consists in computing the resulting action of each rule (or the fuzzy output) based on the antecedent values.

Typically, a minimum function is used so the strength of the rule is assigned to the value of the weakest or least true antecedent. When more than one rule applies to the same specific action, it is common to use a maximum function to assign the strongest (or most-true) consequence. With this action the inferred conclusions from all fuzzy rules and overlapping conditions are combined into a final conclusion.

To understand this step, it is effective to think of a panel of experts, where each panel member stands for a fuzzy rule. Each expert will determine a result and a confidence measure between zero and one to say to what degree the fuzzy rule is satisfied. To come up with a combined resulting action, the responses will be weighted by a confidence measure.

Finally, if it is necessary, the fuzzy conclusion can be defuzzified to obtain a crisp conclusion. This is needed to understand the meaning of a fuzzy action and also to resolve conflicts between competing actions. Defuzzication resolves vagueness and conflicting issues.

There are two techniques to obtain the defuzzified value: calculate the Mean of Maximum (MOM) by calculating the average of all variable values with maximum membership degrees; or calculate the Center of Gravity (COG) or centroid method to calculate the weighted average of a fuzzy set. For our implemented system we selected the COG defuzzification method.

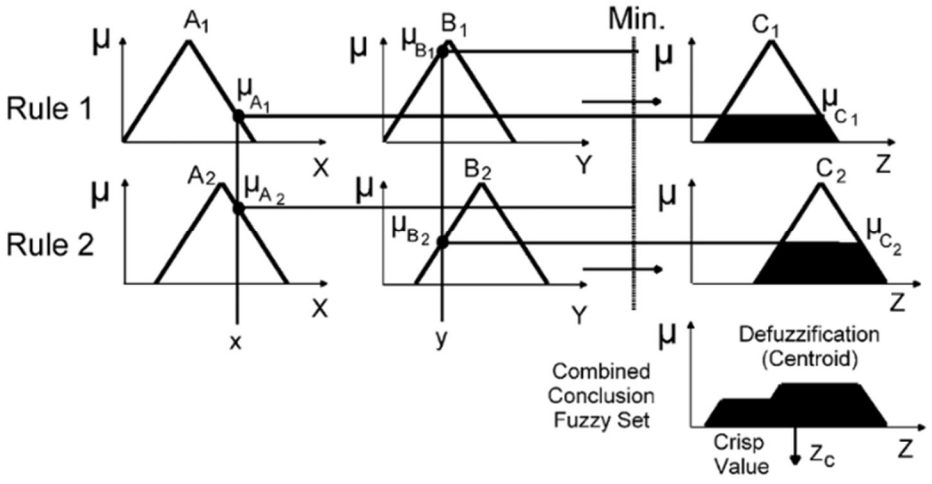


Illustration 2.9: Steps of a Fuzzy Inference System [Rus20].

Because of certain similarities between neural networks and fuzzy logic, researchers investigate ways to combine the two technologies. Such combinations have led to research studies using different levels of integration ranging from internal interaction creating fuzzy neurons,

providing changes in the processing mechanism itself, or as an external fuzzy supervisor. Examples of this will be presented in the next chapter.

CHAPTER 3: LITERATURE REVIEW

According to the latest Artificial Intelligence Index Report, the past 20 years have recorded a dramatic growth in research and development efforts, commercial applications, publications and dissemination activities in the increasingly complex and competitive field of AI [Hai21]. The number of AI journal publications grew by 34.5% in a single year (2019-2020); the number of academic-corporate co-authored publications in the US exploded in the last five years, and the number of patents has seen a big jump in the last three years. State-of-the-art AI technologies show the progress made in many subfields of AI, including the industrialization of computer vision for object recognition and medical image analysis; the rapid emergence of systems using Natural Language Processing (NLP) used in search engines and virtual assistants; the adoption of ML to make significant breakthroughs in healthcare and biology, and combined efforts to accelerate COVID related drug discovery and help with diagnostics during the pandemic.

Such successful results in many instances come at the cost of long training times specially in systems using large datasets and big networks [Cir12, He15]. Although BP is the standard and highly effective algorithm used in DLN, it can suffer from slow training. That is why companies and researchers in diverse fields are investing large amounts of resources to train their systems at a faster rate than ever before. This is important because the faster you can train a system, the more quickly you can evaluate it and update it [Hai21].

Researchers have shown successful reduction in DLN training time by using different approaches: changes in the hardware, preprocessing of the data, optimizing the learning rate or momentum parameters, using different activation functions to simplify operations, or changes in the weight updating procedure.

Some examples of studies related to computing power show improvements by increasing the number of accelerators used—from a couple of hundred to more than four thousand—to cut the training time from several minutes to just seconds [Mlp20]. Other authors report reduced training times by implementing convolutional neural networks using mid-range FPGAs while still maintaining same performance as when implemented on high end GPUs [Ovt15, Lin15]. Other authors indicate they achieved 15 to 30 times faster training times for MLP and CNN networks implemented on a TPU located in a datacenter [Jou17]. The downside of this approach is the increased cost due to complex architectures, expensive technology, or relying on computational power provided by cloud services.

Some studies focus on increasing or decreasing the learning rate parameter to avoid getting stuck (e.g., oscillating on local minima) and slow down convergence by using adaptive techniques to estimate the optimal value of the learning rate [Kol19]. Others use error curve changes to modify the learning rate causing also an improvement in accuracy [Ami18]. Another study achieved training time improvement by adapting a global learning rate based on the error measured during validation [Duf07]. The disadvantage of estimating the learning rate at each iteration of the gradient descent is that it can be computationally expensive and require added memory.

Some researchers have used fuzzy techniques to improve the BP performance by using fuzzy-based activation functions to reduce the number of epochs and the number of neurons [Kar04]. Others use fuzzy control systems to adjust the learning rate, depending on the shape of the error surface, and improve training time dramatically [Ara92]. The use of an adaptive fuzzy approach to control the learning parameters, based on RMS of error surface, obtained a 30% reduction of training time [Ras12]. Additionally, a fuzzy-controlled delta rule can be used to adjust the network weights, according to a parallel coordinate descent method whose parameters are

fuzzy-controlled [Lip94]. Other authors indicate that a network can be trained with a local error signal, using layer-wise loss instead of a global loss function to obtain a system with more biological plausibility [Nok19]. The use of a fuzzy weight adjustment to generalize the BP algorithm in a triple network ensemble [Gax12] has also provided promising results. Using fuzzy logic offers an interesting alternative when sharp boundaries are undesirable, and when dealing with uncertainty.

The proposed research aims at optimizing the learning process used in DLNs, while at the same time maintaining performance accuracy. Specifically, a two-phase fuzzy inference system will be used to modify the Backpropagation (BP) algorithm. A fuzzy decision support system, utilizing a cost function will be created to choose when a fuzzy inference system, composed of learning rules should be invoked, and the process of updating the neuronal weights carried out. If the decision support system determines that this process should be skipped, it will typically be when the network is performing "well enough" and updating the weights will not yield a substantial benefit.

The traditional algorithms calculate error, and proceed with updating the weights, regardless of the error magnitude. That is, traditional algorithms have the error margin set to zero, or close to zero. The problem with this traditional view is that the weights would be updated, even if completing this process will cause minimal change in network, which will ultimately have negligible impact on the overall performance. This study aims at providing optimization by inserting an intelligent agent that will make decisions based on the current performance of the network with the goal of reducing the number of required operations during the training phase by avoiding unnecessary weight modifications and, thereby, speedup the network.

CHAPTER 4: IMPLEMENTATION

After reviewing the trends and strategies that have been used in previous studies, this chapter will cover the details and necessary steps taken towards the implementation of the proposed research idea of inserting an intelligent agent within the training processes used in DL. This agent will infer a decision based on the performance of the network. Specifically, a two-phase fuzzy inference system will be integrated into the backpropagation algorithm to serve as a decision support system, with the aim of speeding up the training process while at the same time maintaining good classification accuracy.

The next sections of this chapter present first, the design choices for the baseline network and the network that will use fuzzy-inference assisted learning; second, a careful analysis of the traditional backpropagation algorithm along with the proposed modifications; and third, how all of these components are integrated into a working model for subsequent implementation in code.

4.1 DESIGN CHOICES FOR NEURAL NETWORKS

There is a need to create two networks to be able to evaluate the performance of the proposed model. The first network is identified as *Trad-BP* and it is considered the baseline network because it uses the *traditional* implementation of the backpropagation algorithm during training. The second network implements the proposed model and it is identified as *FIL*-BP because its training will be assisted by *Fuzzy Inference Learning* for the backpropagation algorithm.

Before the proposed model could be implemented into a program, extensive research was completed, in order to understand the training processes, the key attributes, and parameters that can be optimized to impact the learning algorithm.

4.1.1 Architecture

As described in chapter two, there are many types of networks used to solve different types of problems in DL.

One of the most popular is the CNN, and different versions are used extensively to classify images, cluster them into classes, and perform object recognition for computer vision systems. To do this, CNNs use an architecture that enables the processing of the inputs through many layers of convolution, pooling, and classification operations.

The architecture of a typical CNN is depicted in Figure 4.1. The core structure in the classification process is a fully connected multi-layer network trained using the backpropagation algorithm. Thus, work on improving learning in a multi-layer feedforward network will also impact other DLNN network architectures, such as the CNN.

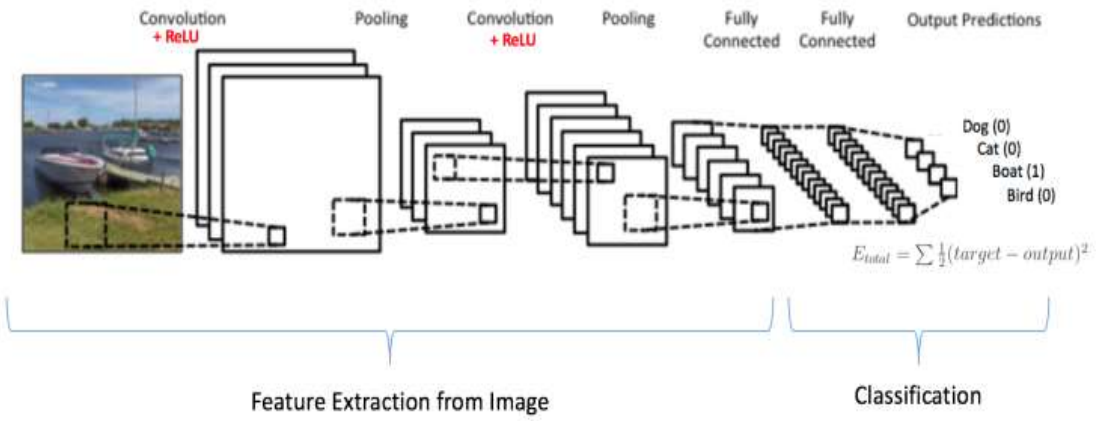


Figure 4.1: Basic building blocks for a simple CNN [Kar16]

The chosen architecture for both networks in this study consists of a feedforward multi-layer neural network with three layers: input, hidden and output as shown in Figure 4.2.

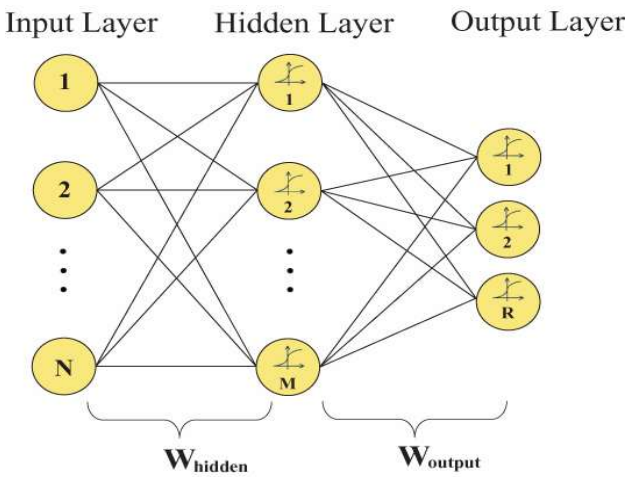


Figure 4.2: Architecture for Feedforward Multi-layer NN with three layers

Each processing neuron has the corresponding number of inputs to accommodate the input vector dependent on the size of the database being used. In addition, each processing neuron receives an extra input of +1 from the 'bias neuron'.

4.1.2 Activation Function

The activation function for the neurons is the binary sigmoid function and it is chosen because it is continuous, differentiable, monotonically increasing, and its derivative is easy to compute, which reduces the computational burden during training. This function, which has a range of (0, 1), and its derivative are expressed by the following equations:

$$f_1(x) = \frac{1}{1 + \exp(- x)} \tag{4.1}$$

$$f_1'(x) = f_1(x)\ [1 - f_1(x)] \tag{4.2}$$

The binary sigmoid function activation function is illustrated in Figure 4.3

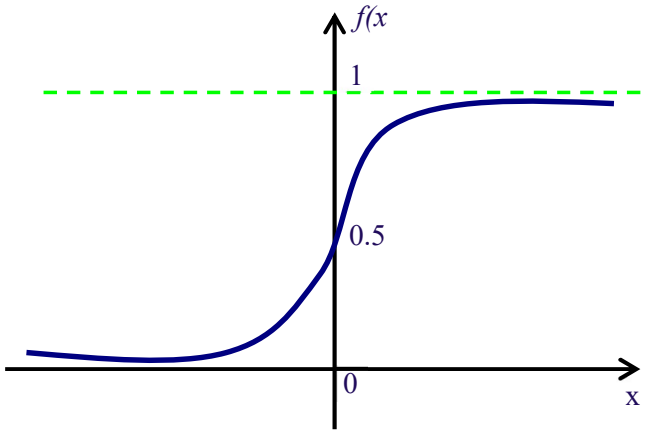


Figure 4.3: Binary Sigmoid Function, Range (0, 1)

4.1.3 Datasets

DLNNs use benchmark datasets in order to evaluate the performance of the learning algorithms. The University of California at Irvine (UCI) provides a Machine Learning Repository hosted by the Center for Machine Learning and Intelligent Systems. This repository is a collection of more than 622 databases that are extensively used by the machine learning community for empirical analysis of machine learning algorithms. Students, educators and researchers all over the world have free access to these datasets [Dua19].

For this study, the following three benchmark datasets were selected for testing the networks in classic classification problems:

- The first one is the XOR dataset. It corresponds to a simple but important classification problem characterized by a small well-defined dataset of four patterns.
- The second one is the Iris dataset. It is perhaps the best-known database to be found in pattern recognition literature. It is based on a classic research paper by Fisher [Fis36] which has been referenced since 1936. It contains 150 patterns divided into 3 different classes.

- The third one is the Connectionist Bench (Vowel recognition) dataset. It contains 990 records across 11 classes. It is a complex data consisting of a three-dimensional array composed of utterances collected from several speakers. The objective is to achieve speaker independent recognition of the eleven steady state vowels of British English.

In classification problems with databases that have a known target output, the most common supervised learning algorithm used is backpropagation.

4.2 BACKPROPAGATION ALGORITHM

Backpropagation algorithm is the standard, most popular, and most effective learning method for training Deep Neural Networks, and is the algorithm used for training the networks in this study. Therefore, this section explains in detail how the algorithm has evolved and the basics of how it works.

One could think of the backpropagation network as an evolved version of the perceptron with multiple layers, different threshold functions, and a more robust and efficient learning rule.

The first learning rule was designed in 1949 by Hebb, was refined by Rochester, Holland, Haibt and Duda in 1956 and by Kohonen and Anderson in 1972. The extended Hebb learning rule for modifying the weight values of a network is given by:

$$w_i(new) = w_i(old) + x_i y \tag{4.3}$$

In 1958, Frank Rossenblatt introduced the adaptive *perceptron* neural network consisting of 3 layers and it was designed to learn to associate a given input to a random output unit. The perceptron learning rule is more powerful than the additive Hebb rule because its iterative learning procedure adjusts the weights when system response is incorrect, such that the system will

converge to the correct weights if in fact a solution exists. The proof is based on classical logic in abstract mathematics, and is known as the "Convergence Theorem." [Rud76]

The perceptron learning rule incorporates the use of the learning rate (α) parameter and it updates the weights according to the following equations:

If input ≠ output,

$$w_i(new) = w_i(old) + \alpha t x_i \tag{4.4}$$

Else

$$w_i(new) = w_i(old) \tag{4.5}$$

Widrow and Hoff developed the delta learning rule by 1960. The delta rule changes weights of the neural connections in order to minimize the difference between the net input to the output unit, y_in, and the target value t resulting in the least mean square error. The aim is to minimize the error over all training patterns [Fau94]. The delta rule for a single output unit adjusting the i^{th} weight is given by:

$$\Delta w_i = \alpha(t - y\_in)x_i \tag{4.6}$$

The delta learning rule led to improved generalization capabilities and it served as precursor to the backpropagation rule used in multilayer networks.

David Rumelhart and David Parker refined the backpropagation (BP) algorithm which compares the result that was obtained by the network with the result that was expected. It uses the gradient descent method to minimize the total squared error with the aim of training a network to be able to respond correctly to given input patterns used for training and be able to give a

reasonable response to new input patterns that are similar, but not identical, to those used in training [Fau94]. This capability is known as generalization.

With the help of the BP algorithm, the hidden neurons are free to construct their own representation of the input information. The connecting set of weights between the input and hidden layers determine when each hidden neuron is active; therefore, the modification of these weights affects how the hidden neuron represents the input being received.

Training a network with the BP algorithm requires the execution of two major phases as illustrated in Figure 4.4:

1. The feedforward propagation phase: each of the input training patterns of the training set are passed or propagated through the network, from the input layer towards the output layer, to obtain the response of the network.
2. Backpropagation phase: based on the network's response, the associated error is calculated and the corresponding adjustment of the weights is backpropagated to adjust all the weights.

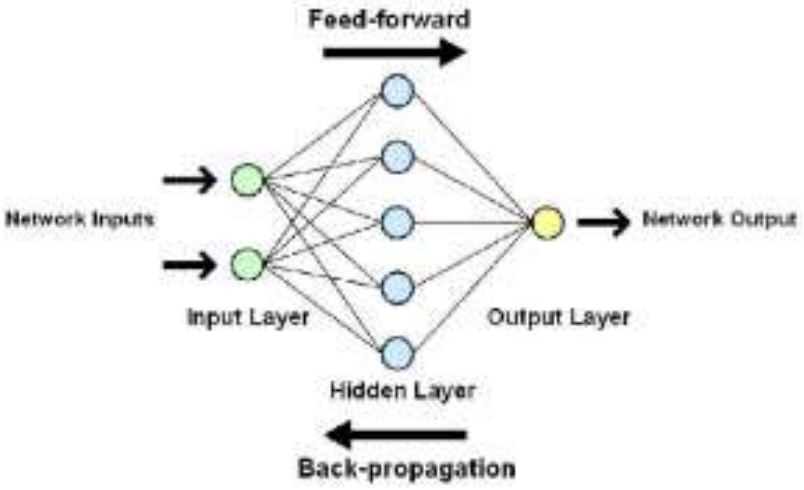


Figure 4.4: Phases of the Backpropagation Algorithm

After training is completed, the response of the network is obtained by completing only the steps in the feedforward phase. This occurs when evaluating the network with a test dataset or when the system is deployed to be used as a classification system.

While training, the weights of each neuron are adjusted so the error between the expected output and the calculated output is reduced. This process requires the computation of the error derivative of the weights. In other words, the network must calculate how the error changes as each weight is increased or decreased by the calculated amount. The process of training is typically repeated for hundreds or thousands of times as the entire training dataset cycles through the network in what is referred to as one epoch.

The mathematical basis for the backpropagation algorithm is the optimization technique known as gradient descent. The gradient of a function gives the direction in which the function increases most rapidly; the negative of the gradient gives the direction in which the function decreases most rapidly [Fau94]. For the NN being used, the function is the error and the variables are the weights of the net.

In other words; since the function is the error, the goal is to minimize such function by following the steepest descent. By changing the values of the weights, the error surface is followed, trying to reach the smallest error (ideally zero).

Some key factors that influence the performance of the backpropagation network are worth mentioning. The initial weights of the neurons should be small, non-zero random values because if they are too large, the network may get stuck at a local minimum, and be prevented from learning.

The learning rate α is also involved in the learning process. Usually, a large value of α yields a faster convergence, but increases the possibility of overshooting, and potentially missing

the solution (weights). A smaller value will decrease the possibility of overshoot, but will slow down the convergence [Au98]. To solve this problem, a momentum term μ is incorporated so that the new change in the weights is dependent on the previous change. Momentum can accelerate the convergence of the backpropagation learning rule, while enhancing the stability of the learning process [Au98].

The learning rate α determines the 'length' of the error gradient vector (purple arrow in Figure 4.5). If α is too large (vector is too long) it risks overshooting the target (global minima) by oscillating. To avoid this, the momentum μ term is added. If traveling in the right direction, μ will contribute to the movement (by making the vector even larger); but if moving in the wrong direction, it will help correct the movement in the opposite direction (by making it shorter).

Determining the optimum values for learning rate and momentum is accomplished through trial and error. The effect that these key terms have on the learning process is illustrated in Figure 4.5 below.

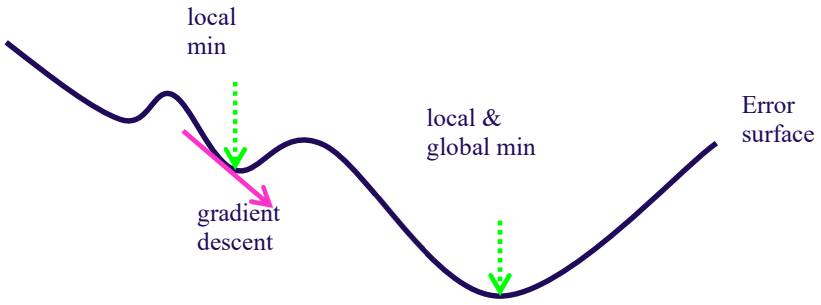


Figure 4.5: Gradient descent moving along the error surface

Now the complete backpropagation training algorithm steps can be summarized as follows:

The neural network has N input terminals, M neurons in a single hidden layer and R neurons in the output layer. There are P pairs of training data:

$$\{(\mathbf{x}_1, \mathbf{t}_1), (\mathbf{x}_2, \mathbf{t}_2), ..., (\mathbf{x}_p, \mathbf{t}_p)\}, \text{ where}$$

$$\mathbf{x}_i \; is \; (\text{Nxl}), \; \mathbf{t}_i \; is \; (\text{Rxl}), \; i = 1,2,\ldots,\text{P}. \tag{4.7}$$

Note that $\mathbf{x}_{\text{aug-p}}$ denotes the augmented vector, defined as

$$\mathbf{x}_{aug-p} = \begin{bmatrix} +1 \\ \mathbf{x}_p \end{bmatrix}, \; for \; p = 1,2,\ldots,P \tag{4.8}$$

k denotes the training step (iteration) and p denotes the current training pattern within the training cycle.

Step 1: $0 < \alpha < 1.0$ and E_{MAX} are chosen.

Step 2: Weights are initialized to small random values.

For the output layer, $\mathbf{W}_{\text{out}}=[w_{ij}]$, where $\mathbf{W}_{\text{out}}$ is of size R x (M+1) (4.9)

For the hidden layer, $\mathbf{W}_{\text{hid}}=[w_{ij}]$, where $\mathbf{W}_{\text{hid}}$ is of size M x (N+1) (4.10)

$$k \leftarrow 1, \; p \leftarrow 1, \; E \leftarrow 0 \tag{4.11}$$

Step 3: The training cycle begins here. Input $\mathbf{x}_{\text{aug-p}}$ is presented and output computed during a forward pass:

$$y_i \leftarrow f(\mathbf{w}_{\text{hid-i}} \, \mathbf{x}_{\text{aug-p}}) \qquad \text{for i=1,2,…,M} \tag{4.12}$$

$$O_i \leftarrow f(\mathbf{w}_{\text{out-i}} \, \mathbf{y}_i) \qquad \text{for i=1,2,…,R} \tag{4.13}$$

where $\mathbf{w}_{\text{hid-i}}$ is a row vector (specifically, the i^{th} row of $\mathbf{W}_{\text{hid}}$), and

$\mathbf{w}_{\text{out-i}}$ is a row vector (specifically, the i^{th} row of $\mathbf{W}_{\text{out}}$).

Step 4: Error and δ's are computed:

$$E \leftarrow \tfrac{1}{2} (t_j - o_j)^2 + E \tag{4.14}$$

For the output layer,

$$\delta_{out\text{-}j} = e_j \cdot f'(v_j) \tag{4.15}$$

$$= (t_j - o_j) \cdot o_j \cdot (1 - o_j) \qquad \text{for } j=1,2,\ldots,R \tag{4.16}$$

(for the binary sigmoid function)

For the hidden layer,

$$\delta_{hid\text{-}j} = \left(\sum_{all\ k} \delta_{out-k} \cdot w_{kj} \right) \cdot o_j \cdot (1 - o_j) \qquad \text{for } j=1,2,\ldots,M \tag{4.17}$$

(for the binary sigmoid function)

Step 5: Weights are updated:

For the output layer,

$$\mathbf{w}_{out\text{-}j} \leftarrow \mu \cdot \mathbf{w}_{out\text{-}j} + \alpha \cdot \delta_{out\text{-}j} \cdot \mathbf{y} \qquad \text{for } j=1,2,\ldots,R \tag{4.18}$$

For the hidden layer,

$$\mathbf{w}_{hid\text{-}j} \leftarrow \mu \cdot \mathbf{w}_{hid\text{-}j} + \alpha \cdot \delta_{hid\text{-}j} \cdot \mathbf{x}_{aug\text{-}p} \qquad \text{for } j=1,2,\ldots,M \tag{4.19}$$

Step 6: If $p < P$,

then $p \leftarrow p + 1$, $k \leftarrow k + 1$, and go to Step 3. (4.20)

Otherwise, go to Step 7.

Step 7: The training cycle (epoch) is completed.

$$\text{If } E \leq E_{MAX} \tag{4.21}$$

then terminate the training session and output weights, *k*, and E.

Otherwise,

$$E \leftarrow 0, p \leftarrow 1, \tag{4.22}$$

and enter a new training cycle by going to Step 3.

It is important to note the following:

- The forward phase constitutes the completion of all the operations listed in equations 4.12, 4.13 and 4.14. To be concise, in the next chapter during the analysis of the results, this set of operations is referred to as completing a *forward function call*.
- The backpropagation phase constitutes the completion of all the operations listed in equations 4.15 through 4.19 as described above. Similarly, for brevity when analyzing the results, this set of operations is referred to as completing a *BP function call.*

Steps three, four and five will be repeated in a cycle as the network processes every single pattern in the training dataset. When all the patterns in the dataset have been processed (an epoch has been completed), *Step* 7 is executed to check if the training session can be terminated due to system convergence. If the tolerable error (E_{MAX}) threshold has not been reached, a new epoch is started and the cycle of steps three, four and five begins again until the system converges or until the maximum number of epochs has been reached (training stopping condition). It is not unusual for a training session to require thousands of epochs before a system can converge.

In addition, the optimum learning rate and momentum parameters are typically determined by trial and error. This means that a whole new training session is required for each new setting of the learning parameters.

When considering that DL applications usually use datasets with millions of records being processed by complex architectures consisting of multiple layers, multiple neurons and millions of trainable parameters that need to be updated, it is easy to understand why they suffer from slow training. For example, when training the AlexNet CNN—composed of 8 layers with more than 60 million parameters and using a training set of 14 million images classified into 20,000 categories—each training *session* required between five to six days to complete.

The aim of this research study is to optimize the training procedure by speeding up the process through the use of the proposed model which, with the help of an intelligent agent, searches for opportunities and determines when the backpropagation phase can be skipped and thereby effectively eliminating the need to calculate derivatives of the error for each neuron in the output and hidden layer, as well as eliminating the need to update each of the corresponding weights accordingly.

The next section describes the design of the intelligent agent that uses a fuzzy inference system to assist in the learning process. Please be aware that the backpropagation training steps described here will be referenced when mentioning the modifications made by the proposed model.

4.3 INTEGRATION OF FUZZY-INFERENCE LEARNING INTO THE BP ALGORITHM

As described in chapter two, fuzzy logic is a powerful, yet straightforward, intelligent technique that incorporates and implements human knowledge and strategies into solving problems in the areas of control and decision making.

The power of fuzzy-based systems is that they have the ability to reach conclusions and generate corresponding actions in a way similar to humans. Instead of relying on sharp boundaries, the control in a fuzzy engine is based on fuzzy rules or heuristics that use imprecise data.

For this study, a two-phase fuzzy inference system (FIS) will be designed and inserted within the BP algorithm for two purposes. First, it serves as a decision support system in making the determination of *when* it is appropriate for the network to interrupt the training steps and skip the backpropagation phase, to avoid the process of updating the neuronal weights. Second, it will provide control of the skipping parameter itself by evoking the learning rules and inferring the corresponding action based on the current performance of the network. If the decision support system determines that the backpropagation phase should be skipped, it will be when the network is performing "well enough" and updating the weights will not yield a substantial benefit.

Recall that the traditional algorithm calculates the error, and proceeds with updating the weights, regardless of the error magnitude. The problem with this traditional implementation is that the weights are updated, even if completing this process causes minimal change in the network, which will ultimately have negligible impact on the overall performance.

The FIL-BP network will take advantage of the inferred conclusions provided by the intelligent agent to effectively and efficiently reduce the number of BP function calls completed during the training procedure and, thereby, speedup the network training process.

The first level of integration of the fuzzy agent with the BP training algorithm comes at the end of the forward phase. As each pattern in the training dataset completes the forward pass through the network, the respective total local error (e_j) gets accumulated while executing the operation given by Equation 4.14. The total local error across all output units is:

$$e_j = \tfrac{1}{2} (t_j - o_j)^2 + e_j \tag{4.23}$$

The fuzzy system will take this value, compare it to a skipping threshold (θ) and will only allow the training process to continue for that particular pattern if the total squared error is greater than the threshold level:

$$\theta > e_j \tag{4.24}$$

In other words, if the network's response to a particular pattern causes a total error to be greater than the skipping threshold, it means the network must learn from that pattern, therefore it should proceed with the backpropagation phase to make adjustments to the weights accordingly.

Initially, the skipping threshold is set equal to the network tolerance used to determine when the system converges. The logic here is that if a pattern already meets the network's tolerable error, there is no need to modify the weights to reduce this local error even further, hence, that particular pattern doesn't need to contribute in the adjustment of the network's current knowledge (contained in the values of the weights).

The second integration comes at the end of each epoch and before the BP algorithm determines if the training session should be terminated (refer to Equation 4.21). At the end of each epoch, the mean square error (MSE) of the current epoch (Ep_{MSE}) at time t is stored:

$$Ep_{MSE}(t) = E \tag{4.25}$$

At this point, the fuzzy inference system will control the skipping threshold based on the current performance of the network. In other words, the $Ep_{MSE}(t)$ will serve as the first input to the fuzzy inference system (FIS), as given by

$$FISin1 = Ep_{MSE}(t) \tag{4.26}$$

The second input to the FIS will be the Change in the system Error (CE) which is calculated by the difference between the values of the system error during the current epoch and the system error at the previous epoch as given by:

$$CE = Ep_{MSE}(t) - Ep_{MSE}(t\text{-}1) \tag{4.27}$$

$$FISin2 = CE \tag{4.28}$$

At this point, FISin1 and FISin2 will go through the fuzzification process where the fuzzy agent will calculate the degree of membership of each input by referencing the corresponding trapezoidal membership functions. These functions are designed and tuned based on careful analysis of the general behavior of the error and the change in error inputs. Figures 4.6 and 4.7 show some examples (in a zoomed-in view) of how FISin1 and FISin2 can change during training.

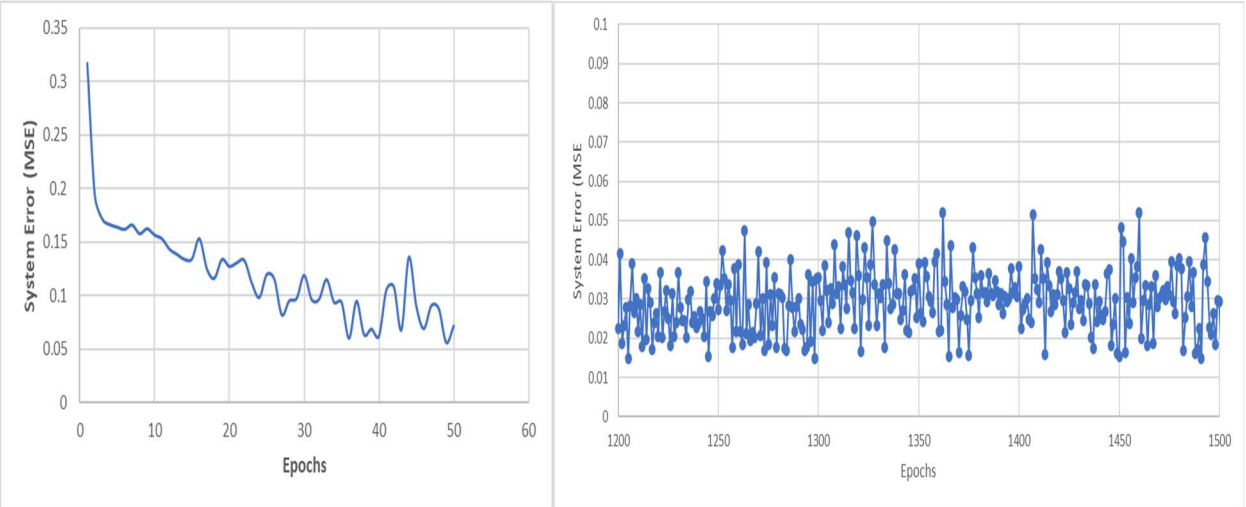


Figure 4.6: Example showing how FISin1 changes during training procedure

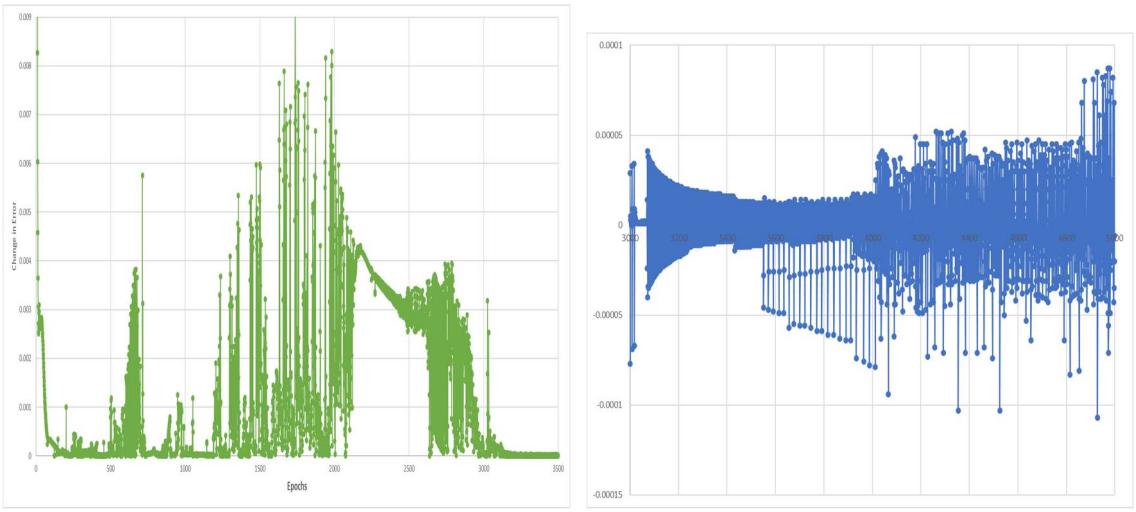


Figure 4.7: Example showing how FISin2 changes during training procedure

The designed membership functions are illustrated in Figure 4.8 and Figure 4.9 respectively. The fuzzy sets used are SM (small), MD (medium), LG (large) and XL (extra-large).

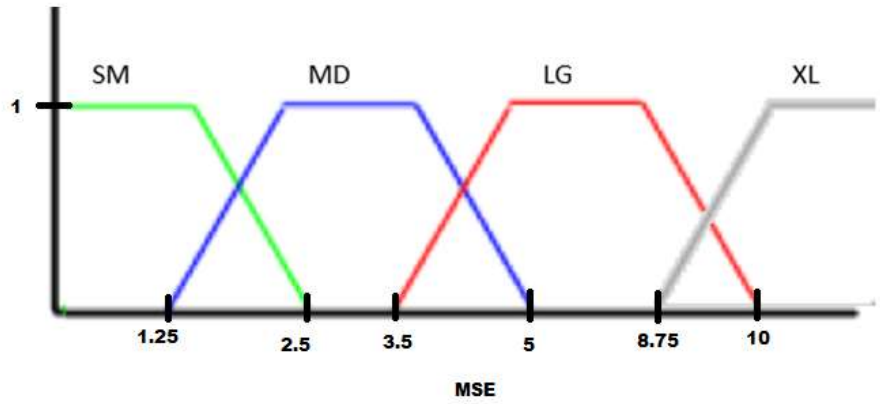


Figure 4.8: Membership functions for FISin1

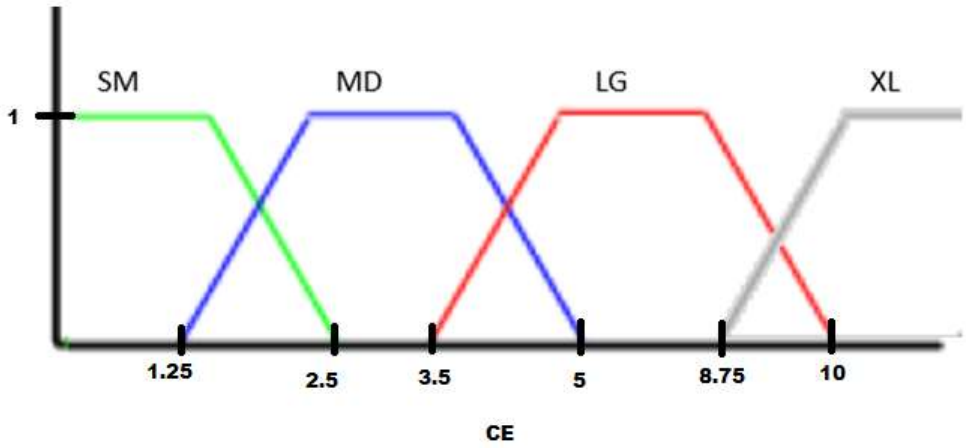


Figure 4.9: Membership functions for FISin2

Figure 4.10 shows the designed output membership functions for controlling the Skip-Threshold (ST). The fuzzy sets are ZE (zero), DS (decrease small) and DB (decrease big)

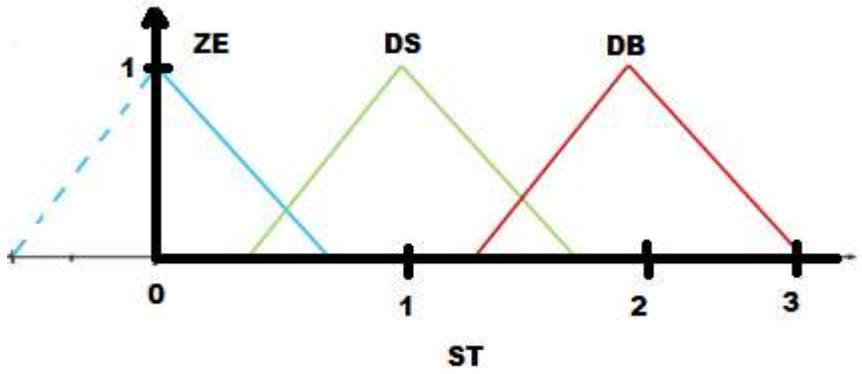


Figure 4.10: Membership functions for ST

As explained in Chapter two, the fuzzification process will calculate the degree of membership by using these functions to map each system input into one or more degrees of membership. Once this is completed, the rules in the rule base are evaluated by combining degrees of membership to form output strengths.

The designed rules for the system use the linguistic terms described for the fuzzy sets. The complete rule base is listed here:

Rule 1: If (MSE is SM) and (CE is SM) then (ST is DB)

Rule 2: If (MSE is SM) and (CE is MD) then (ST is DS)

Rule 3: If (MSE is SM) and (CE is LG) then (ST is ZE)

Rule 4: If (MSE is SM) and (CE is XL) then (ST is ZE)

Rule 5: If (MSE is MD) and (CE is SM) then (ST is DS)

Rule 6: If (MSE is MD) and (CE is MD) then (ST is ZE)

Rule 7: If (MSE is MD) and (CE is LG) then (ST is ZE)

Rule 8: If (MSE is MD) and (CE is XL) then (ST is ZE)

Rule 9: If (MSE is LG) and (CE is SM) then (ST is ZE)

Rule 10: If (MSE is LG) and (CE is MD) then (ST is ZE)

Rule 11: If (MSE is LG) and (CE is LG) then (ST is ZE)

Rule 12: If (MSE is LG) and (CE is XL) then (ST is DS)

Rule 13: If (MSE is XL) and (CE is SM) then (ST is ZE)

Rule 14: If (MSE is XL) and (CE is MD) then (ST is ZE)

Rule 15: If (MSE is XL) and (CE is LG) then (ST is DS)

Rule 16: If (MSE is XL) and (CE is XL) then (ST is DB)

These rules are summarized on the fuzzy matrix show on Table 4.1 below.

Table 4.1: Rule Base Fuzzy Matrix

ST		CE			
		SM	MD	LG	XL
MSE	SM	DB	DS	ZE	ZE
	MD	DS	ZE	ZE	ZE
	LG	ZE	ZE	ZE	DS
	XL	ZE	ZE	DS	DB

Depending on the values of the inputs, the system might trigger multiple fuzzy rules so the results are combined by using a minimum function so the strength of the rule is assigned to the value of the weakest or least true antecedent.

When more than one rule applies to the same specific output action, the system uses a maximum function to assign the strongest (or most-true) consequence. With this action, the inferred conclusions from all fuzzy rules and overlapping conditions are combined into a final conclusion, illustrated by the following example.

Suppose the following conditions exist: Rule 1: if A & B then Z & X and also Rule 2: if C & D then Z & Y

Strength of Rule 1 = min (A, B) (4.29)

Strength of Rule 2 = min (C, D) (4.30)

X= strength of Rule 1 (4.31)

Y = strength of Rule 2 (4.32)

Z = max (Strength of Rule 1, Strength of Rule 2) (4.33)

= max (min (A, B), min (C, D) (4.34)

Finally, the fuzzy conclusion is defuzzified using the center of gravity method which consists of several steps.

First, a centroid point on the x-axis is determined for each output membership function.

$$Centroid = a + \frac{d-a}{2} \quad (4.35)$$

where a and d are the end points of the trapezoid functions.

After this, the membership functions are limited in height by the applied rule strength, and the areas of the membership functions are calculated:

$$Area = h\ \frac{base+top}{2} \quad (4.36)$$

where h is the corresponding strength.

At this point, the defuzzified output is derived from the weighted average of the x-axis centroid points and the computed areas, with the areas serving as the weights:

$$Deffuzified\ value = \frac{\sum\ sum\ of\ products\ (\ area * centroid)}{\sum sum\ of\ areas} \quad (4.37)$$

The resulting value is the crisp value that will be used to reduce ST.

4.4 INTEGRATION OF COMPONENTS INTO A C PROGRAM

Although there are existing tools that allow the user to implement DLN and fuzzy inference systems by specifying the corresponding parameters, for this study, those options did not provide the flexibility required to modify the training algorithm by embedding the fuzzy agent within the BP algorithm itself to control the decision making and when necessary, interrupt the training processes when such action is inferred. Using the tools available would have required to modify the framework's source code in order to implement the interface and achieve the actions of the proposed model. Therefore, the framework for this study was completely designed and created using the C programming language. Both the Trad-BP and FIL-BP networks are implemented by the developed program.

The baseline version of the program has a main function that sequentially makes several secondary function calls to perform the different phases of the backpropagation algorithm. The basic code is modified to additionally incorporate the intelligent agent that will infer a decision based on the performance of the network. As illustrated on Figure 4.11, the fuzzy inference system completes three major function calls for fuzzification of inputs, evaluation of the rules and defuzzification of the output which controls the value that determines when to execute the backpropagation function call, or when to skip it.

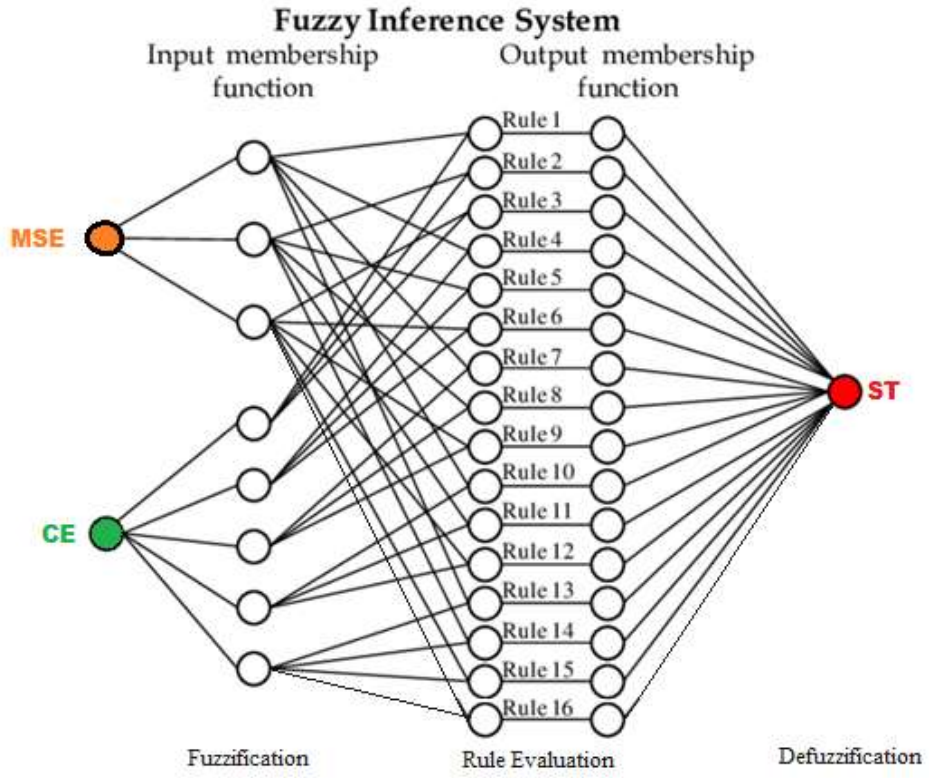


Figure 4.11 Illustration of the major processes of the fuzzy inference system

These tasks are preceded by an initialization routine that sets up the data structures used for the input and output membership functions; obtains the definition of the membership function variable sets; and sets up the rule base for the decision-making process.

The flow of the program code that implements the proposed model can be observed in Illustration 4.1. The listing of the program code is presented in Appendix A.

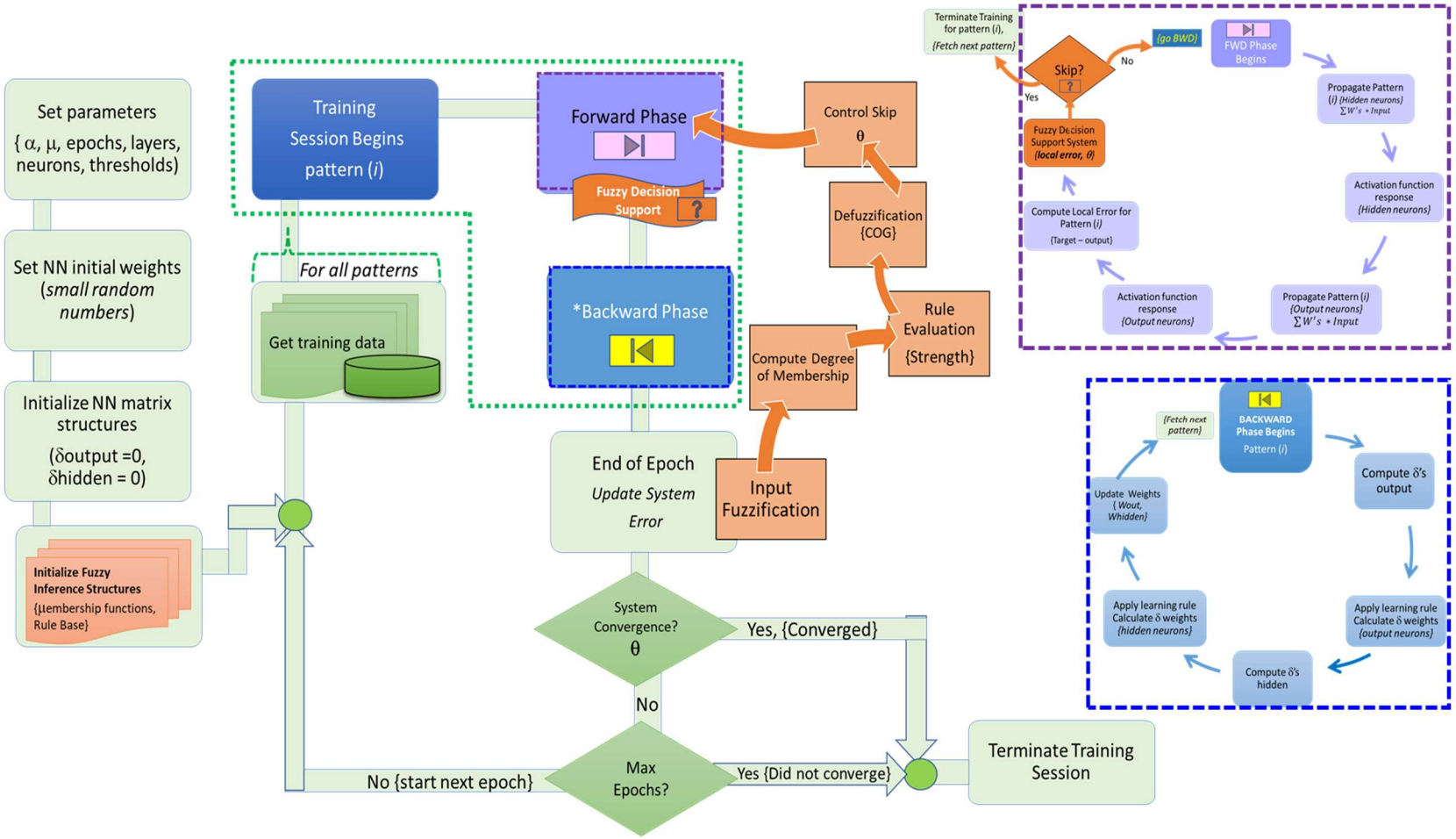


Illustration 4.1: Flow of program code that implements Trad-BP and FIL-BP model

CHAPTER 5: RESULTS

To evaluate the performance of the proposed algorithm, this chapter presents the results obtained after training the two networks described in the previous chapter. The network that utilizes the traditional backpropagation algorithm is referred as *Trad-BP* and the network that uses the proposed fuzzy-inference assisted learning modification to the backpropagation algorithm is referred as *FIL-BP*.

The strategy is to utilize three different databases with different levels of complexity, each more complex than the previous one. This will show the effectiveness of the proposed model versus the traditional implementation when each faces problems of different characteristics and posing diverse challenges. Each type of problem is discussed in the next sections.

5.1 SMALL-SCALE DATABASE: CLASSIC XOR PROBLEM

The XOR (exclusive or) problem is a simple but important problem that any network must solve in order to show the ability to successfully classify between two well-defined non-linearly-separable classes. This problem is typically selected as the first case in any study, due to its simplicity, minimal inputs, well-defined classes, and small data set size.

In this classic problem, the database consists of four training input/output patterns representing the response of the XOR function for two binary inputs (*true* is represented with a one, and *false* is represented with a zero). The behavior of the XOR function is easily captured by the following description: if only one of the two input values is “true”, then the output is “true”; in all other cases, the output is false. Table 5.1 summarizes these four cases in binary notation and it serves as the representation of the full training dataset used by the two networks.

Table 5.1: XOR dataset

A	B	XOR (A, B)
0	0	0
0	1	1
1	0	1
1	1	0

With only four training patterns, this dataset poses a non-linearly separable problem because the solution requires two planes in order to separate between the two classes. This can be visualized in Figure 5.1.

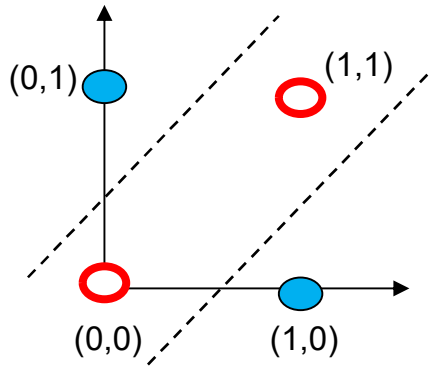


Figure 5.1: Visualization of XOR Problem

The response of the network is explained as follows: if the output neuron fires, it means that the response to the given pattern is true, and if it doesn't fire it means the response is false.

Both networks are implemented using the same architecture which consists of two input neurons, two hidden neurons, and one output neuron. The networks are trained using the BP algorithm under different values for the learning rate (α) and momentum (μ) parameters.

As mentioned on chapter 4, the weight parameters for the neurons collectively hold the knowledge of the network. These weights can't be calculated using analytical methods, weights

are found using the training algorithm. The BP algorithm follows the error function to change the values of the weights by repeatedly using gradient descent to take steps towards reaching the smallest error. The challenge is that the solution space can have many local and global minima; for this reason, α and μ parameters are added to help optimize this search for the solution.

Gradient descent updates the values of the weights by finding the derivative of the error function with respect to the weights. The derivative is the update or the change to be made on the weight parameters, and because this derivative term is being multiplied by the learning rate, the size of the step or the amount of change in the weights is controlled by α. Therefore, the value of the α has an effect on how quickly or slowly a network learns about the problem.

Choosing a proper α can be difficult [Ben12]. When α is too small, training is not only slower, but it may also lead to becoming stuck at a local minimum. When α is *too large*, it risks overshooting the target as gradient descent can inadvertently increase the training error rather than decrease it. This obviously hinders convergence and can cause the loss function to oscillate around the minimum or even diverge. In the worst case, the weight updates become too large which may cause weights to explode (overflow).

In addition to α, μ is added to the calculation of the update by multiplying the value of the previous weight. If traveling in the right direction, μ will contribute to the movement; but if moving in the wrong direction, it will help correct the movement in the opposite direction. In other words, μ increases the update for those dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. As a result, faster convergence and reduced oscillation can be achieved.

Choosing the values of α and μ parameters is not straightforward and their optimum values are usually found by trial and error. For the networks used here, these values are dynamically changed before the start of a new training session according to the (α,μ) set described by equation 5.1:

$$(\alpha,\mu)=\begin{bmatrix}(0.1,0.1) & (0.1,0.2) & \cdots & \cdots & (0.1,0.8)\\ (0.2,0.1) & \cdots & \cdots & \cdots & \cdots\\ \vdots & \cdots & \cdots & \cdots & \cdots\\ \vdots & \cdots & \cdots & \cdots & \cdots\\ (0.8,0.1) & (0.8,0.2) & \cdots & \cdots & (0.8,0.8)\end{bmatrix} \tag{5.1}$$

The duration of each training session is set to have a maximum number of iterations equal to 5,000 epochs. Duration can be less if the system converges to an accumulated system error that is under the 0.01 tolerance.

Because the XOR dataset consists of only the four essential training patterns, the instinctual expectation is that the FIL-BP network will have very little chance to find and skip a pattern that doesn't intrinsically contribute towards reaching the solution. This behavior is confirmed by the parallel results when comparing both networks.

For all of the combinations of the learning rate and momentum parameters, both Trad-BP and FIL-BP networks have the same performance. Figure 5.2 shows that both networks reach the same minimum system error, converging below the error tolerance on almost all the cases. Because the performance of both networks is identical, the graph uses two different colors and visual representations of the data in order to avoid confusion about which data point belongs to which network's performance.

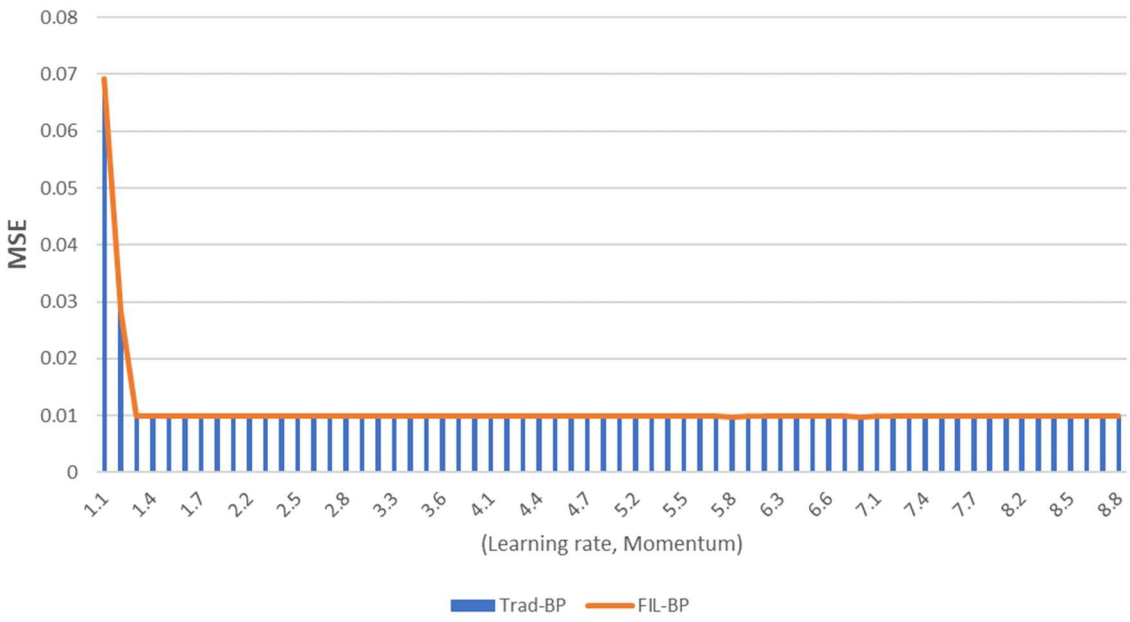


Figure 5.2: System error after training with XOR dataset

For the XOR problem, the training set is identical to the testing set. It is expected that the system will have a perfect classification accuracy. This is confirmed for all the parameter settings that yield to convergence. Figure 5.3 shows the identical 100% classification accuracy of both systems. In this graph, again, two different visual representations of the data are used, in order to clarify the different networks' values and facilitate comparison.

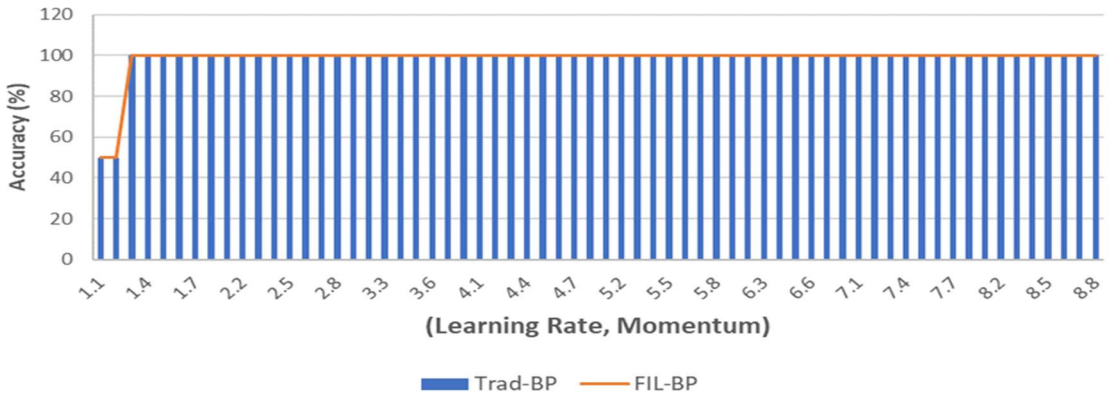


Figure 5.3: Classification accuracy for the XOR testing dataset

Figure 5.4 shows that both networks required the same number of epochs under all of the possible combinations of the learning parameters. Notice that the fastest training occurred at $(\alpha,\mu)= (0.8, 0.8)$ when both systems converged after 192 epochs.

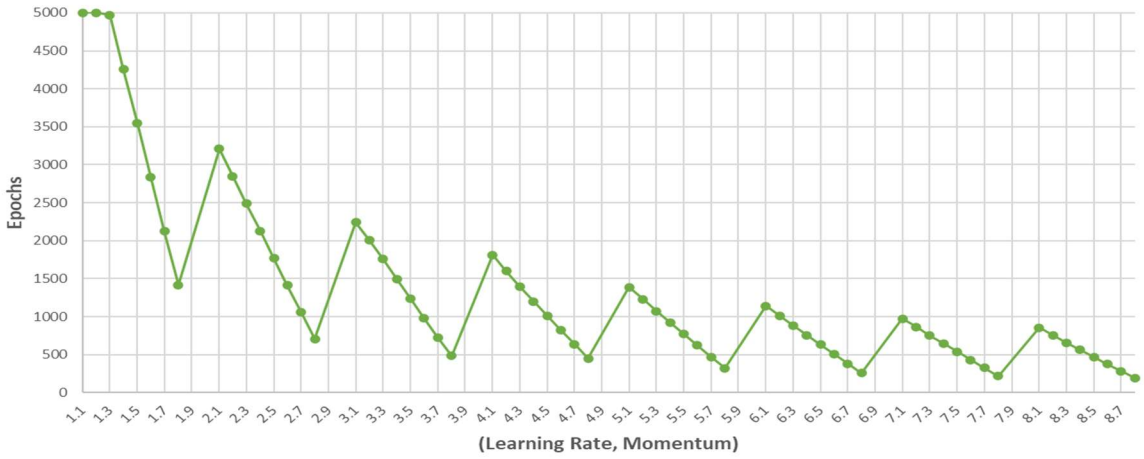


Figure 5.4: Epochs required to complete training with XOR database

There is only one small difference in terms of shortening the training effort between both networks. There was only one occasion when the FIL-BP network was actually able to avoid doing the backpropagation of two training patterns during the 453 epochs completed. Although this doesn't constitute a major savings when comparing both systems, Trad-BP makes 1812 BP function calls while FIL-BP provides a savings of only two by making 1810 function calls. Table 5.2 summarizes the performance of both networks when this occurs at $(\alpha,\mu)= (0.4, 0.8)$.

Table 5.2: XOR training performance at $(\alpha,\mu)= (0.4, 0.8)$

	LEARNING RATE α	MOMENTUM μ	Epochs completed	System Error (MSE)	FORWARD PHASE (# of function calls)	BACKWARD PHASE (# of BP function calls)	SAVINGS in BP (# of BP function calls skipped)	SAVINGS PERCENTAGE (% of BP skipped)	HITS	Classification Accuracy (%)
Trad-BP	0.4	0.8	453	0.00982	1812	1812	0	0	4	100
FIL-BP	0.4	0.8	453	0.00982	1812	1810	2	0.110375276	4	100

Although the savings might not be significant to consider the use of FIL-BP as beneficial in this case, it is important to highlight that even when skipping some patterns, neither the minimization of the error nor the generalization are impacted. In other words, both networks' performances are identical as they both reach the same minimum system error, and both have the same classification accuracy.

Table 5.3 provides a summary of the performance of both networks while training with the XOR dataset across all combinations of the learning parameter and momentum parameters.

Table 5.3: Summary of training with XOR dataset

		TRAINING *(Dataset size= 4)*			PERFORMANCE			EVALUATION *(Dataset size= 4)*	
		EPOCHS completed	FORWARD PHASE (# of function calls)	BACKWARD PHASE (# of BP function calls)	SAVINGS in BP (# of BP function calls skipped)	SAVINGS PERCENTAGE (% of BP skipped)	SYSTEM ERROR (MSE)	HITS	Classification Accuracy (%)
Trad-BP	***MAX***	5000	20,000	20,000	0	0	0.069264	4	100
	MIN	192	768	768	0	0	0.009758	2	50
	AVG	1342.89	5,372	5,372	0	0	0.011162	3.94	98.44
FIL-BP	***MAX***	5000	20,000	20,000	**2**	**0.110**	0.069264	4	100
	MIN	192	768	768	0	0	0.009758	2	50
	AVG	1342.89	5,372	5,372	0.03	0.00	0.011162	3.94	98.44

The complete table showing the comparison between both networks for each of the training settings is found in Appendix B.

5.2 Well-behaved Medium-size Database – Iris Classification Problem

The iris classification problem utilizes one of the best-known databases used in pattern recognition literature since it was collected by R. A. Fisher [Fis36]. It is a standard benchmark problem for testing different classification methods [Hua02]; therefore, it is used as a second, more complex testbench in assessing the performance of both Trad-BP and FIL-BP networks.

This problem consists of classifying three classes of iris plants based on four flower attributes. The data set was obtained from the University of California at Irvine (UCI) Machine Learning Repository [Dua19].

The database contains 150 mixed patterns: 50 Iris-Setosa, 50 Iris-Versicolor and 50 Iris-Virginica (see Figure 5.5).

Figure 5.5: Three Classes: Iris-Setosa, Iris Versicolor and Iris-Virginica

Each pattern has 4 flower attribute values: sepal length, sepal width, petal length, and petal width. The general characteristics of the database are summarized in Table 5.4.

Table 5.4: Iris Data Characteristics

		MIN	MAX	Average
Sepal	length	4.3	7.9	5.84
	width	2	4.4	3.05
Petal	length	1	6.9	3.76
	width	0.1	2.5	1.2

Both networks are implemented using the same architecture which consists of four input neurons, four hidden neurons, and three output neurons. As in the previous testbench, the networks are trained by dynamically changing the different values for the learning rate (α) and momentum (μ) parameters before the start of each new training session according to the (α,μ) set described by equation 5.1.

The duration of each training session is set to have a maximum number of iterations of 5,000 epochs. Duration can, of course, be less if the system converges to an accumulated system error that is under the 0.01 tolerance prior to the epoch limit.

The iris database is considered "well-behaved" by experts in the field because the training set has excellent and equal representation of all the three classes, and the attributes are inherently so meaningful that most networks with this data set typically obtain high classification rates. One of the classes is linearly separable from the other two, but the other two classes are *not* linearly separable from each other.

Among the three problems being used for this study, this database can be considered medium in size. The total size of this database is 150 records distributed equally among the three classes (50 patterns for each category). The training set has 90 samples and the testing set has 60 samples.

To provide a better perspective of the performance of the FIL-BP network, the analysis begins with the savings achieved, compared to the Trad-BP, while training across all the learning rate and momentum combinations.

As described in Chapter 4, each network goes through a forward phase and a backwards phase while training. The Trad-BP always completes the *same* amount of forward function calls (during propagation) as BP functions calls (during backpropagation), effectively providing zero savings. This is because each pattern is passed forwards and backwards throughout the duration of the training, as per the original learning algorithm, with no efficiency, i.e., opportunity of skipping.

For instance, when Trad-BP takes 5,000 epochs for training, each of the 90 patterns in the training set are passed forward 5,000 times and backwards 5,000 times. This means that there are

450,000 forward function calls **and** another 450,000 BP function calls (*90 patterns times 5,000 epochs equal 450,000 function calls for each phase*) for a total of 900,000 functions calls by the end of *each* training session (a new training session is initiated for each of the 64 learning rate and momentum combinations).

The efficiency of the proposed model is calculated by comparing the number of BP function calls it would have made (explicitly given by the actual number of forward calls required by the number of epochs completed) to the number of BP function calls *skipped* by the FIL-BP network. Thus, this figure is referred to as "savings" in the following paragraph, since the work is *saved* by skipping function calls.

Since the savings come from the backwards phase only, Figure 5.6 shows the total number of BP functions calls made by each of the networks while training. When comparing one network to the other, it becomes apparent that the FIL-BP completes a considerably smaller number of function calls under every single combination of the learning rate and momentum parameters, providing a substantial amount of savings.

A complementary visualization of these results is presented in Figure 5.7. In this graph the y-axis shows the percentage of BP function calls skipped by each network. Notice Trad-BP network provides zero percent of savings as it never skips backpropagation of any pattern during training. FIL-BP provides savings throughout the settings, reaching an astonishing maximum of up to 83% of BP functions skipped.

Given the large number of patterns skipped during training, it is crucial to analyze the impact this has on the network's performance, determined by the system error and generalization capacity during evaluation.

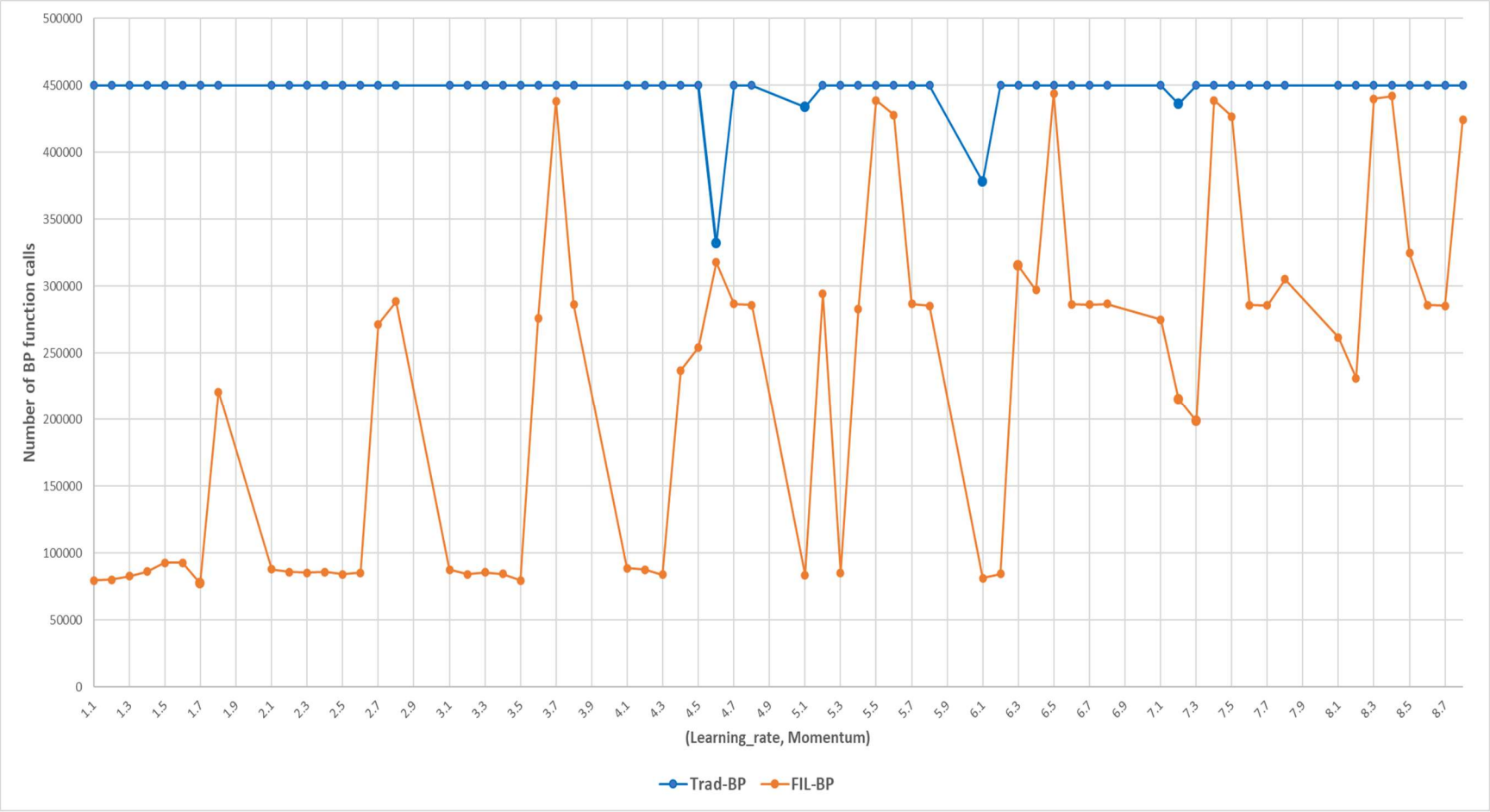


Figure 5.6: Number of BP Function Calls, during training with the iris dataset

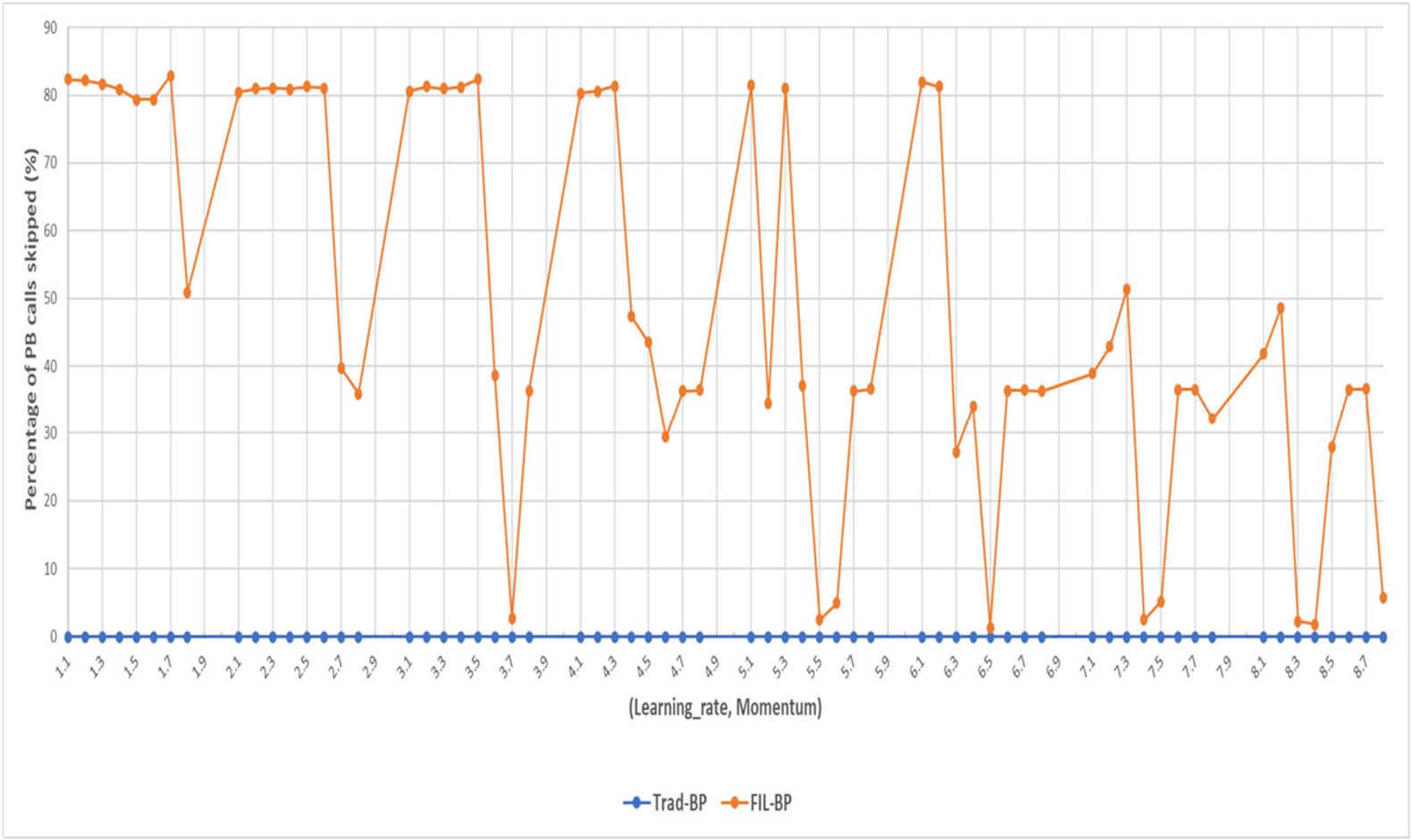


Figure 5.7: Percentage of BP function calls skipped, training with the iris dataset

Surprisingly, when comparing the final system error of FIL-BP to that obtained by Trad-BP, the performance of one almost mirrors the other, being very similar in most cases. In most cases, the performance difference between the two systems is insignificantly small, sometimes even favoring the use of the FIL-BP. There are a few instances where the difference is much larger, but occurs in only a few cases, with both of the networks showing at least once instance of outperforming the other. Figure 5.8 shows the comparison of the minimized system error achieved by both networks.

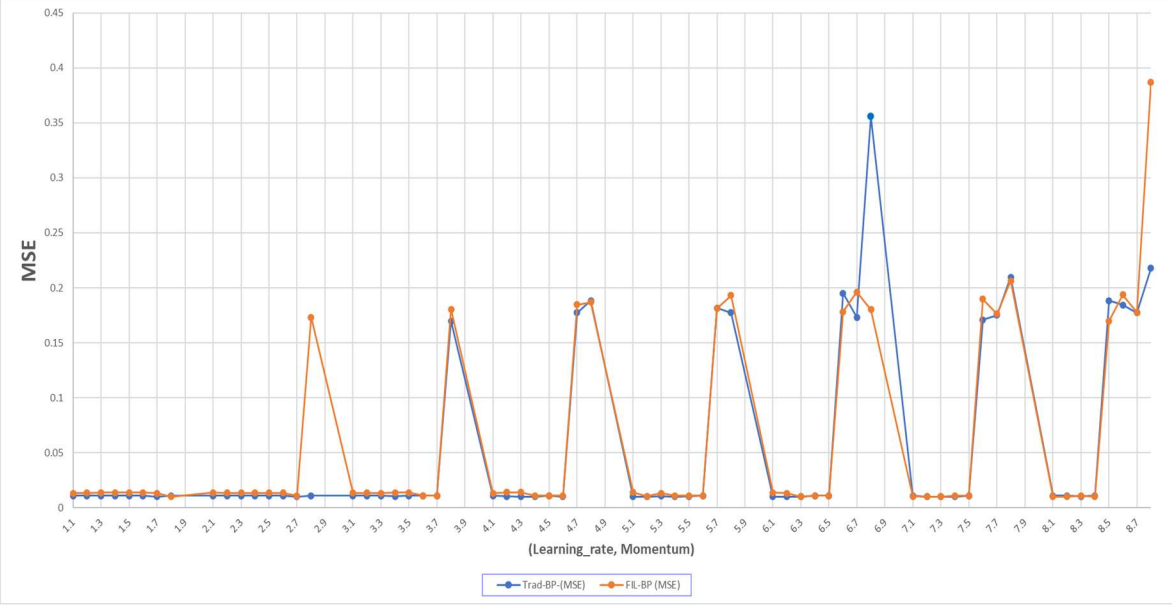


Figure 5.8: System error of Trad-BP and FIL-BP after completing training

Remarkably, the FIL-BP network does not suffer any negative impact on its classification accuracy by being more efficient in its training. Figure 5.9 show the classification accuracy of both networks across all learning parameter combinations. Both networks produce almost the mirror image of each other, with FIL-BP having better on some occasions. There are a few instances with a much larger difference, but it occurs in favor of both networks at different points.

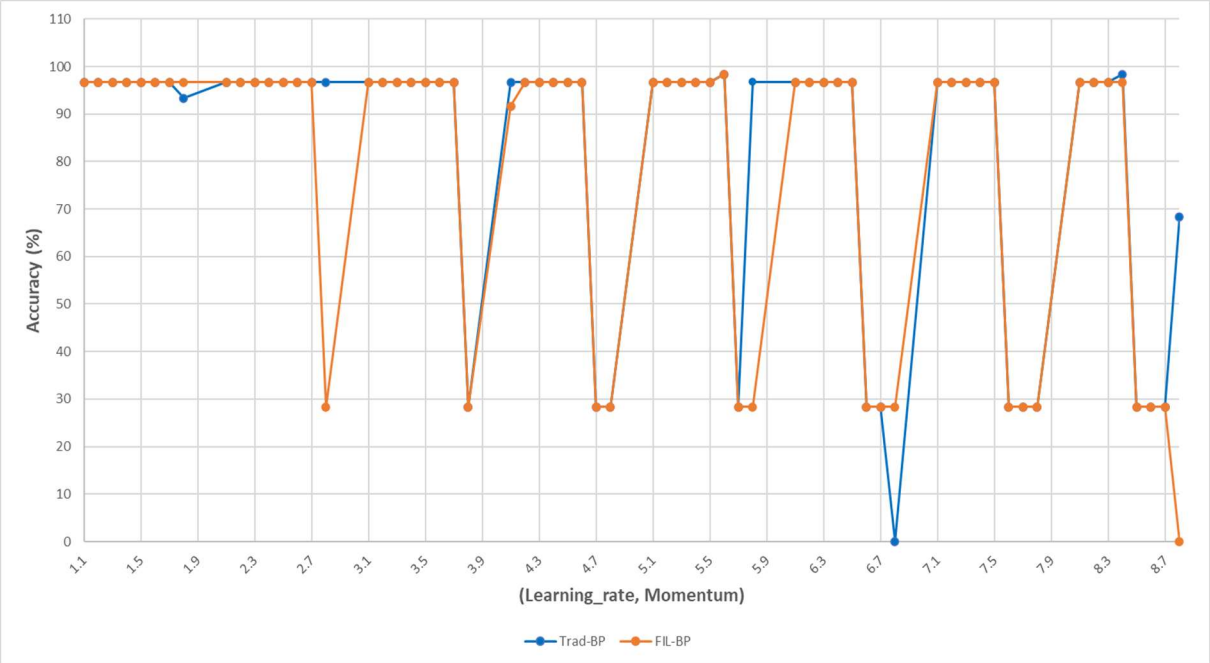


Figure 5.9: Classification of Trad-BP and FIL-BP for iris database

To take a closer look at the results provided by FIL-BP, the following figures compare the training process of both networks as they converged under the (α,μ)= (0.7, 0.2) parameters. Figure 5.10 and 5.11 show how, during training, the Trad-BP and FIL-BP networks iteratively reduced the system error.

An interesting artifact is observed in Figure 5.10, approximately around epoch 1000, and highlighted by an arrow: the Trad-BP model appears to be momentarily stuck at a possible local minimum. Figure 5.11 demonstrates that this situation seems to be completely avoided by the FIL-BP.

Leftward pointing arrows have been added to Figures 5.10 and 5.11, to show when both systems reach the same initially stabilizing error, inching closer to convergence. FIL-BP reaches that milestone by epoch 3400, while the Trad-BP requires 3143 epochs to begin stabilization. Although the Trad-BP reaches the milestone first, the FIL-BP converges faster!

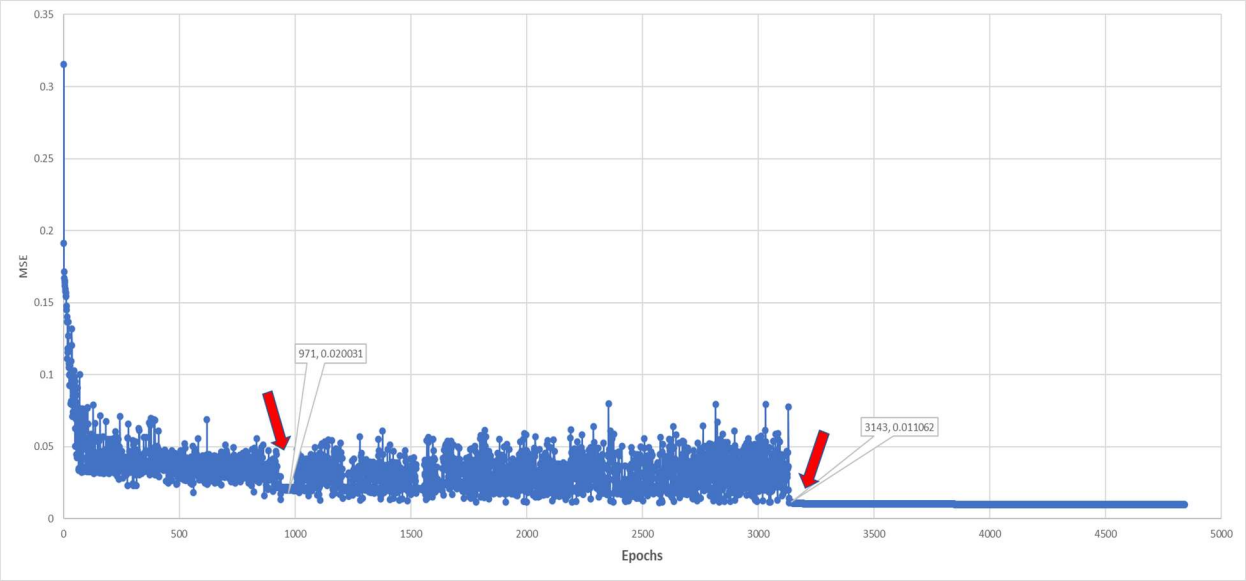


Figure 5.10: System error reduction by Trad-BP for (α,μ)= (0.7, 0.2)

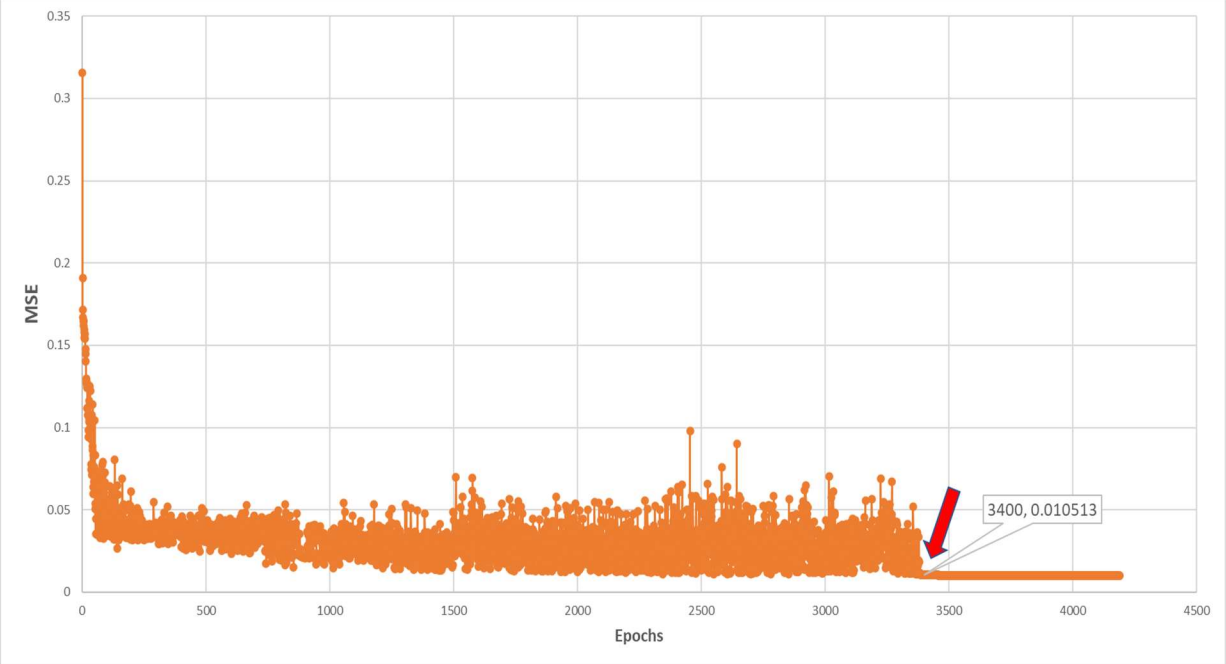


Figure 5.11: System error reduction by Trad-BP for (α,μ)= (0.7, 0.2)

Figure 5.12 superimposes the error reduction of both networks on the same graph. This shows that FIL-BP converges to the 0.01 tolerable error mark much sooner. While Trad-BP reaches convergence by epoch 4842, FIL-BP converges and stops the training by epoch 4188, a difference of 654 epochs. This shows a savings already, if number of epochs was the metric. (It

can be shown that number of epochs is a good metric, as each epoch is related to execution time, so the savings in epochs is directly proportional to greater efficiency in execution time of the training phase.) The charts on the next pages will show the numerical savings in terms of BP function calls skipped for (α,μ)= (0.7, 0.2) and other combinations.

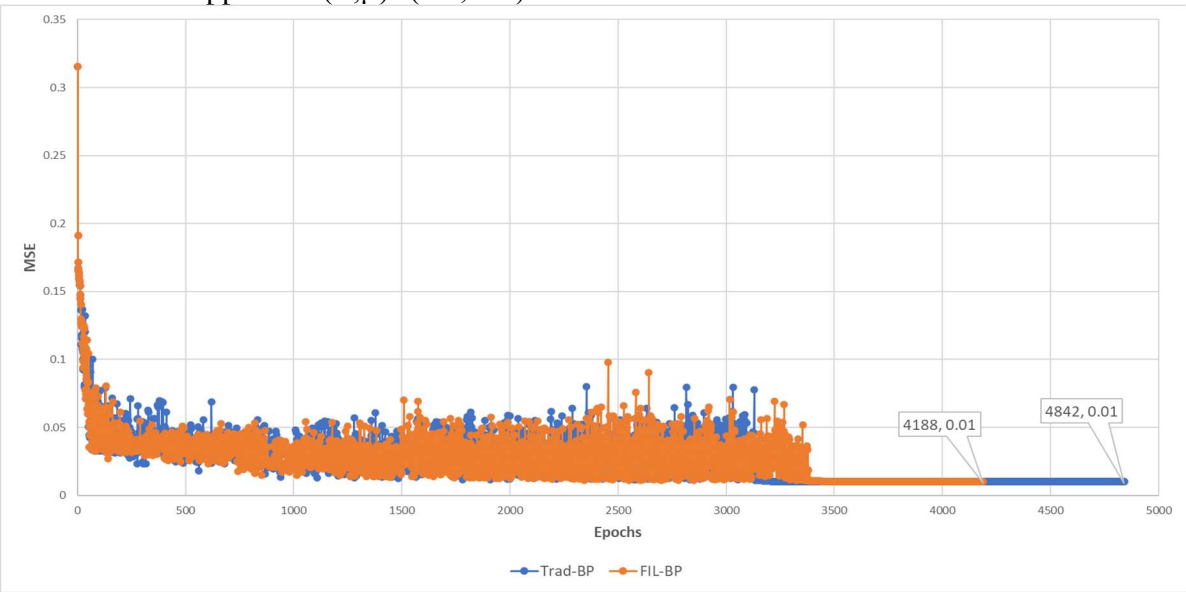


Figure 5.12: System error reduction by Trad-BP and FIL-BP for (α,μ)= (0.7, 0.2)

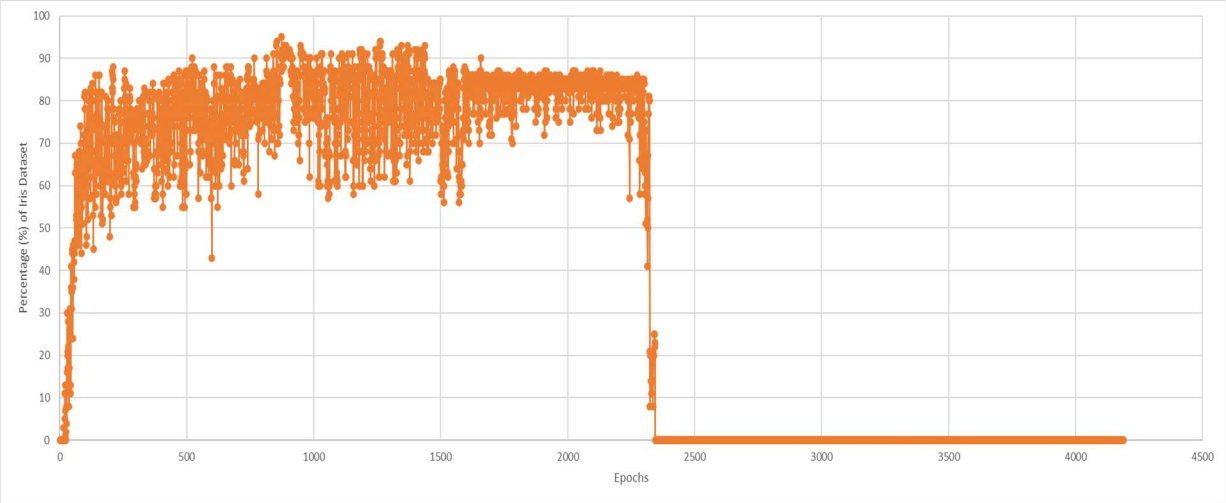
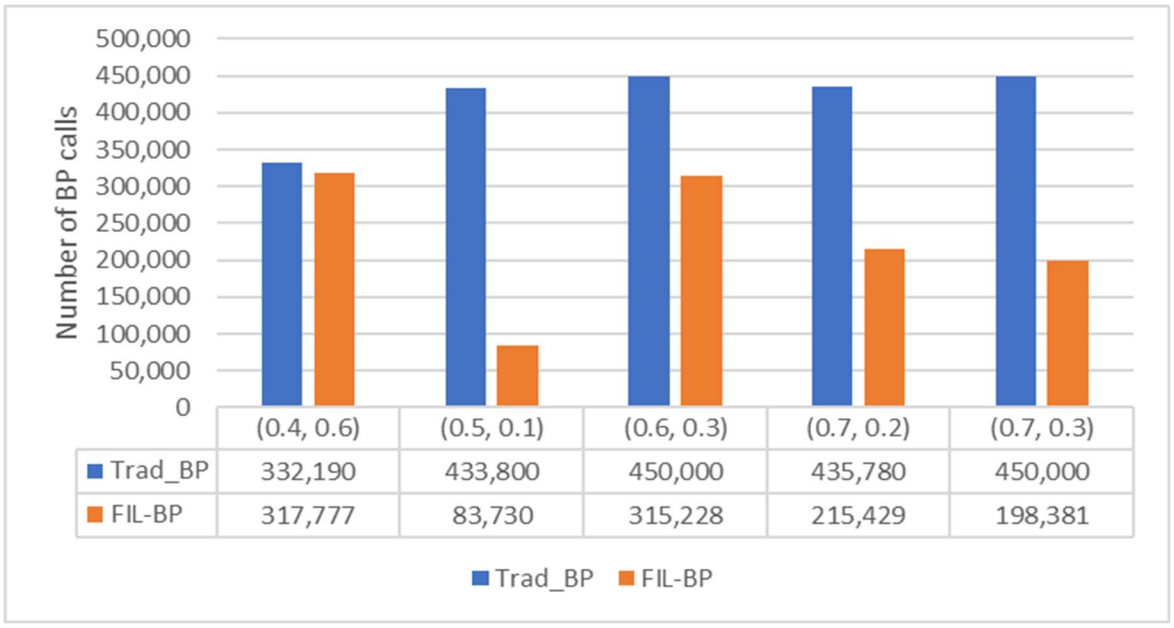


Figure 5.13: Percentage of iris database skipped by FIL-BP for (α,μ)= (0.7, 0.2)

Figure 5.13 shows the percentage of the database that is skipped while training for (α,μ)= (0.7, 0.2). Notice it reaches points when it skips 95% of the database!

To complete the comparative analysis of the savings FIL-BP provides for the iris dataset, five instances where either network reached convergence were selected. The next figures show the comparison for training sessions under the following pairs of learning parameters: (α,μ)= {(0.4, 0.6), (0.5, 0.1), (0.6,0.3), (0.7,0.2) and (0.7,0.3)}.

Figure 5.14 shows visually and numerically the number of BP calls made by both networks while training. In every instance, FIL-BP requires a smaller number of calls, which means the system will provide the corresponding savings on execution time during training.



	(0.4, 0.6)	(0.5, 0.1)	(0.6, 0.3)	(0.7, 0.2)	(0.7, 0.3)
Trad_BP	332,190	433,800	450,000	435,780	450,000
FIL-BP	317,777	83,730	315,228	215,429	198,381

Figure 5.14: Number of BP function calls during different training sessions

Figure 5.15 shows there is no negative impact on the final system error due to the skipped patterns. The dotted red line points out the 0.01 error mark (convergence point). Notice the difference in magnitude for the biggest error in pair (α,μ)= (0.5, 0.1) could be considered acceptable given the savings of 82% this same setting provides.

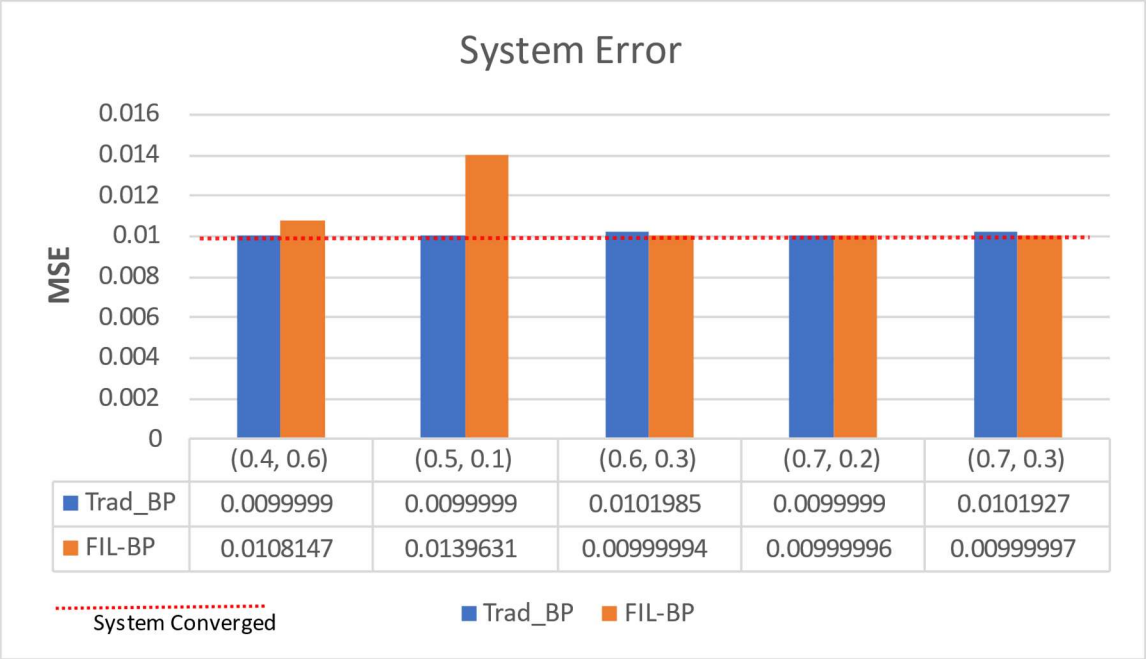


Figure 5.15: Final system error during different training sessions

Finally, for all of these instances, the classification of both systems is exactly the same. In other words, even though FIL-BP skipped many patterns, it did not affect its ability to generalize with the same accuracy as Trad-BP. These results are show in Figure 5.16.

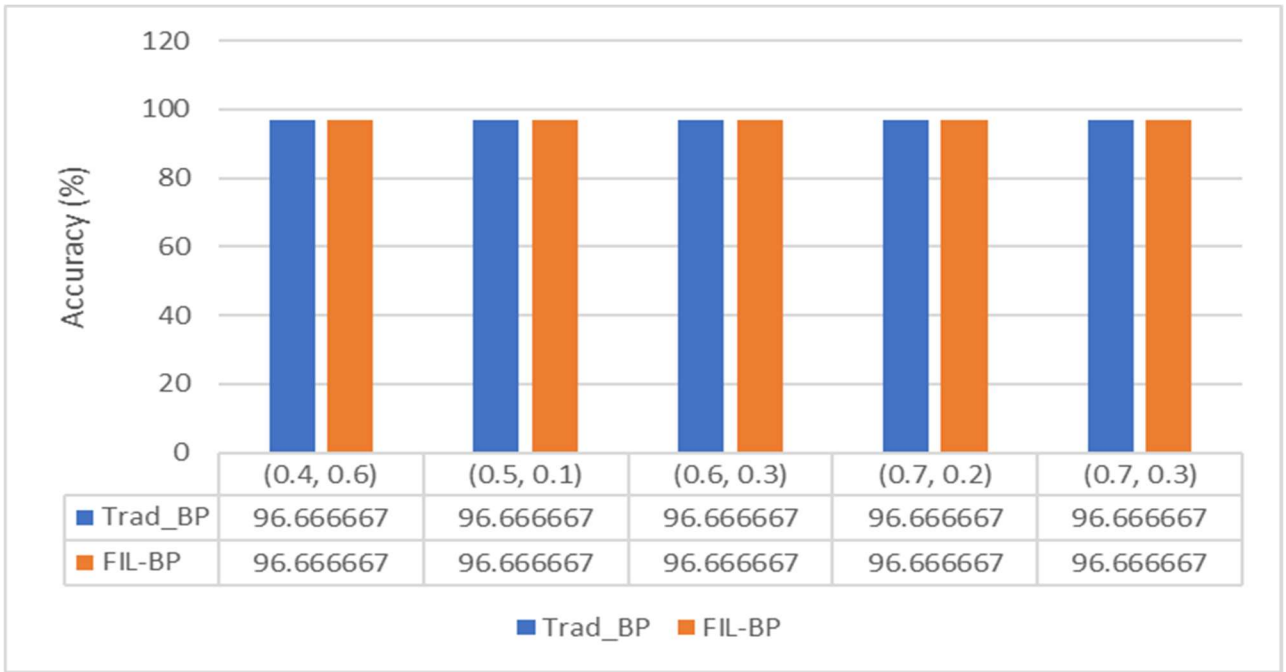


Figure 5.16: Classification after completing different training sessions

In an attempt to summarize the overall satisfactory performance of FIL-BP over Trad-BP, the maximum, minimum and average values of each category across the 64 rows corresponding to each learning parameter combinations is listed on Table 5.5. The complete table showing the comparison between both networks for each of the learning parameters used while training with the iris database is found in Appendix C.

Table 5.5: Summary of training results for iris database

		TRAINING *(Dataset size= 90)*			PERFORMANCE			EVALUATION *(Dataset size= 60)*	
		EPOCHS completed	FORWARD PHASE (# of function calls)	BACKWARD PHASE (# of BP function calls)	SAVINGS in BP (# of BP function calls skipped)	SAVINGS PERCENTAGE (% of BP skipped)	SYSTEM ERROR (MSE)	HITS	Classification Accuracy (%)
Trad-BP	***MAX***	5000	450,000	450,000	0	0	0.355858	59	98.333333
	MIN	3691	332,190	332,190	0	0	0.01	0	0
	AVG	4961.73	446,556	446,556	0	0	**0.054089**	49.14	**81.90**
FIL-BP	***MAX***	5000	450,000	443,884	372,760	82.836	0.386947	59	98.333333
	MIN	4188	376,920	77,240	6,116	1.359111	0.01	0	0
	AVG	4977.08	447,937	224,083	**223,854**	**49.94**	**0.058380**	47.45	**79.09**

5.3 Badly-behaved Large-size Database – Connectionist Bench (Vowel recognition) Classification Problem

The objective of this classification problem is to be able to recognize the eleven steady state vowels of the British English independent of the speaker utterance. The sounds are for the following eleven words: *hid, hId, hEd, hAd, hYd, had, hOd, hod, hUd, hud, hed.*

The database was collected from 15 different speakers, each saying each of the eleven state vowels six times. The training set has 528 samples, while the testing set has 462 samples which combine into a database of 990 records. As with the previous problem, the database was also obtained from the University of California at Irvine (UCI) Machine Learning Repository [Dua19].

It is important to highlight that this database, collected by Tony Robinson for his dissertation research, falls into a type of classification problem that he, himself, describes as impossible. The goal here is to maximize a less than perfect performance. It should also be noted that this data set is recognized in the historical literature as "badly behaved" due to the relatively low performance that most systems achieve with this data.

Both networks are implemented using the same architecture which consists in this case of 10 input neurons, 10 hidden neurons, and 11 output neurons. As in the previous problem, the networks are trained by dynamically changing the different values for the learning rate (α) and momentum (μ) parameters before the start of each new training session according to the (α,μ) set described by equation 5.1.

The duration of each training session is set to have a maximum number of iterations, equal to 5,000 epochs. This duration could be less if the system actually converges to an accumulated system error that is under the 0.01 tolerance. It should be noted that due to the complexity of the database, the system doesn't converge under these settings. Classification performance would probably benefit from a much higher number of training epochs, larger architecture, and greater computational power. However, the purpose of this study is to use the database under this manageable setting to show the benefits and efficiency that FIS-BP provides, over a traditional BP network, when facing a complex problem.

Under this setup, training for 5,000 epochs with a training dataset of 528 patterns requires the Trad-BP system to pass each pattern 2,640,000 times forward, and another 2,640,000 times backwards for a total of 5,280,000 function calls by the end of each of the 64 training sessions which combine to a total of 337,920,000 functions calls if no skipping is implemented.

Figure 5.17 shows the total number of BP functions calls made by each of the networks while training. Notice that Trad-BP executes 2,640,000 BP function calls for *each* of the 64 learning rate and momentum combinations. Even for a complex problem such as this one, FIL-BP completes a considerably smaller number of function calls under every single combination of the learning rate and momentum parameters providing a substantial amount of savings.

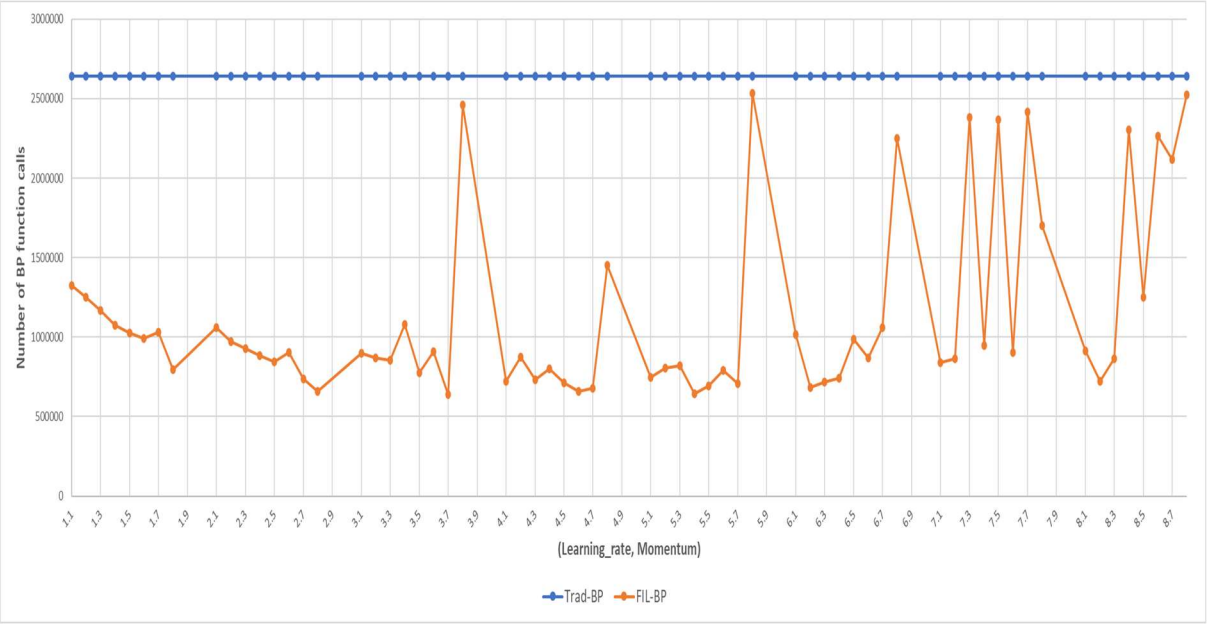


Figure 5.17: Savings provided by FIL-BP when training with vowel dataset

The complementary visualization of these results, depicting savings in percentages, is presented in Figure 5.18. In this graph, the y-axis shows the percentage of BP function calls skipped by each network. Notice Trad-BP network provides zero percent of savings as it never skips backpropagation of any pattern during training. FIL-BP provides extensive savings throughout the settings, reaching an incredible maximum of up to 76% of BP functions skipped.

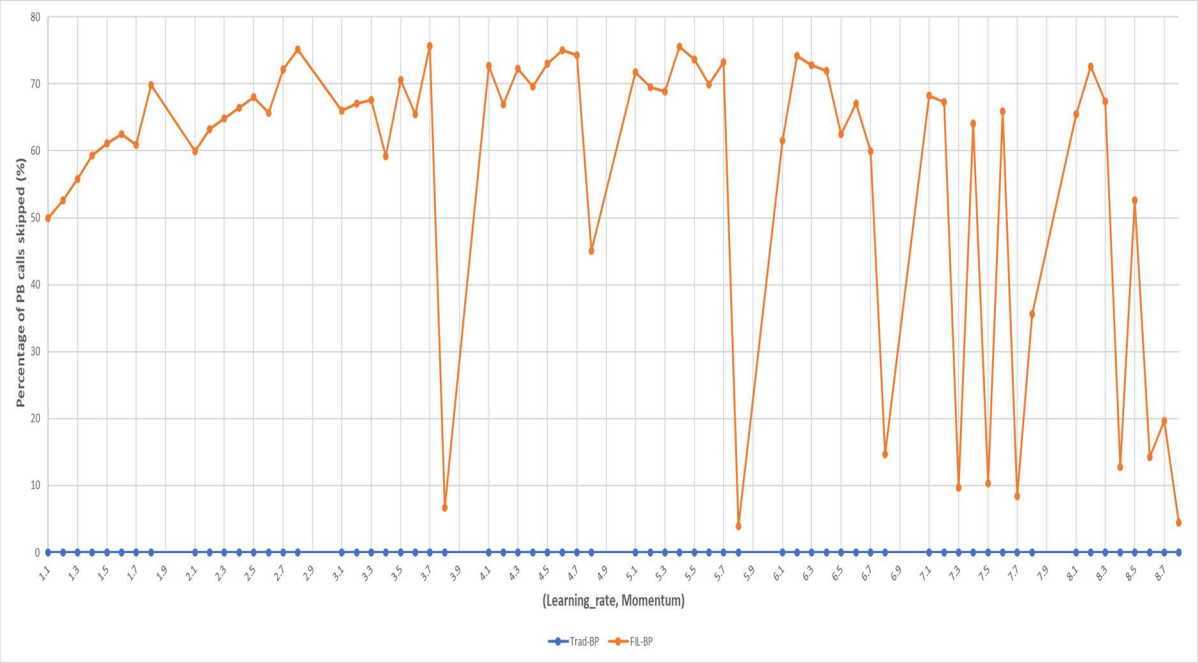


Figure 5.18: Percentage of BP function calls skipped by FIL-BP when training with the vowel dataset

While analyzing the impact of such high efficiency (i.e. skipping BP calls), one should consider what effect this might have on the system error and generalization capability during evaluation. It is very exciting to report that, again, there is *no* significative difference, in either metric, between the Trad-BP and FIL-BP models.

Figure 5.19 shows the error reduction achieved by both networks. Notice the behavior of FIL-BP network again very closely follows that of the Trad-BP which used 100% of the training dataset every time. Remarkably, there are many instances where FIL-BP achieves a better system error reduction than Trad-BP. This might indicate that the FIL-BP doesn't over-train, hence avoiding overfitting.

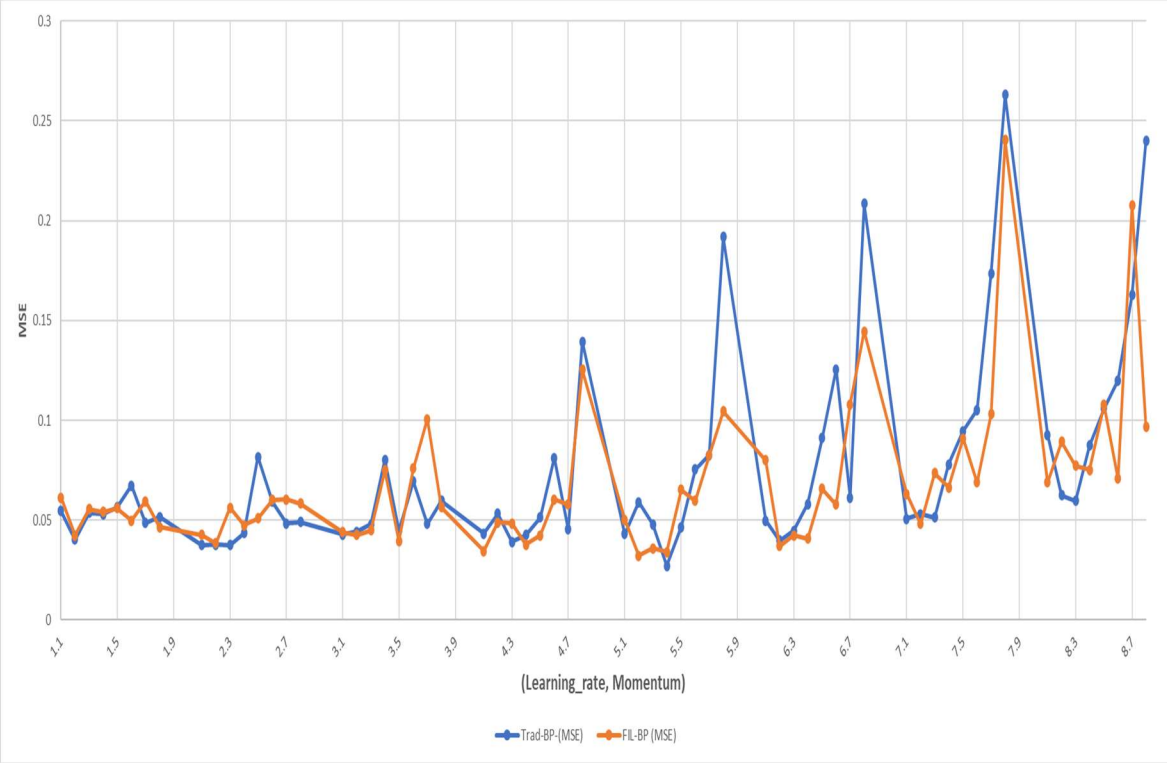


Figure 5.19: System error for Trad-BP and FIL-BP after completing training

FIL-BP doesn't suffer any negative impact on its classification accuracy either. Figure 5.20 shows the classification accuracy of both networks across all learning parameter combinations is very similar and, in some instances, FIL-BP has a better classification rate.

On average, FIL-BP is almost one percent better in its classification effort which seems to confirm the notion that, to a certain degree, the chance of overfitting is diminished.

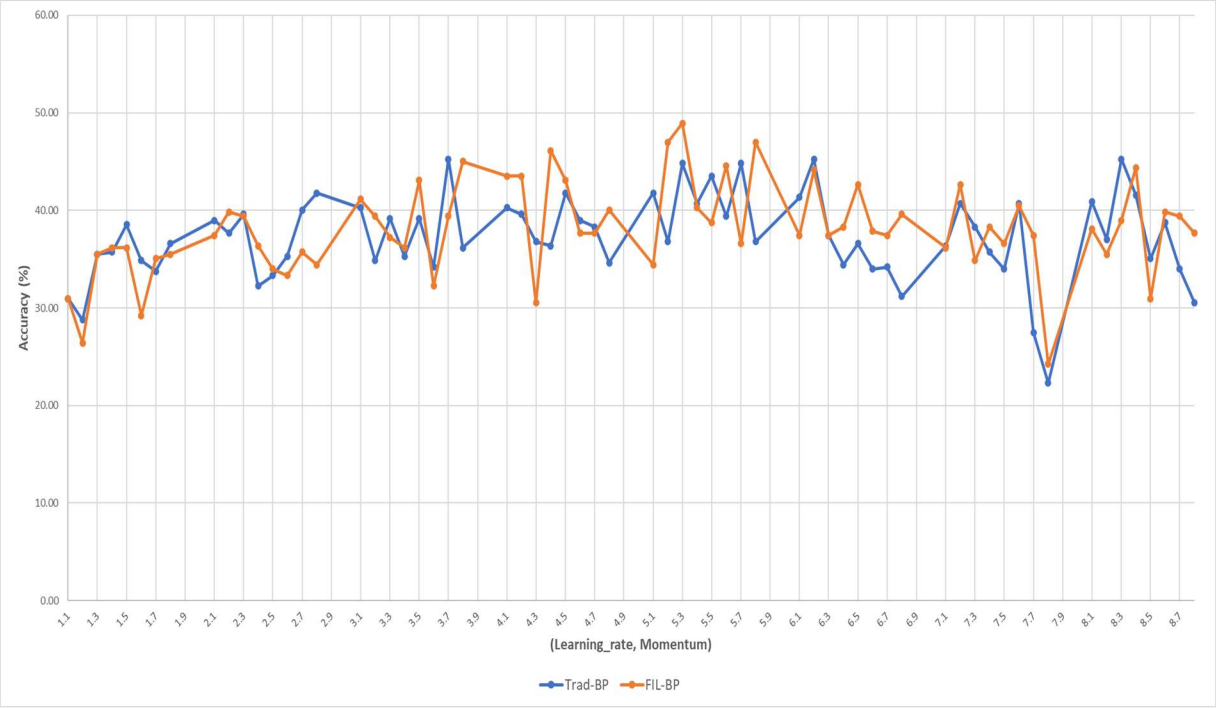


Figure 5.20: Classification accuracy for Trad-BP and FIL-BP

To continue the comparative analysis of the savings FIL-BP provides for the vowel dataset, the four instances showing best performances were selected. The next figures show the comparison for training sessions under the following pairs of learning parameters: $(\alpha,\mu) = \{(0.3, 0.7), (0.5, 0.2), (0.5,0.3)$, and $(0.5,0.4)\}$.

Figure 5.21 and 5.22 show, respectively, the number of BP calls executed and the percentage of calls skipped while training. In every instance, the savings provided by FIL-BP is substantial.

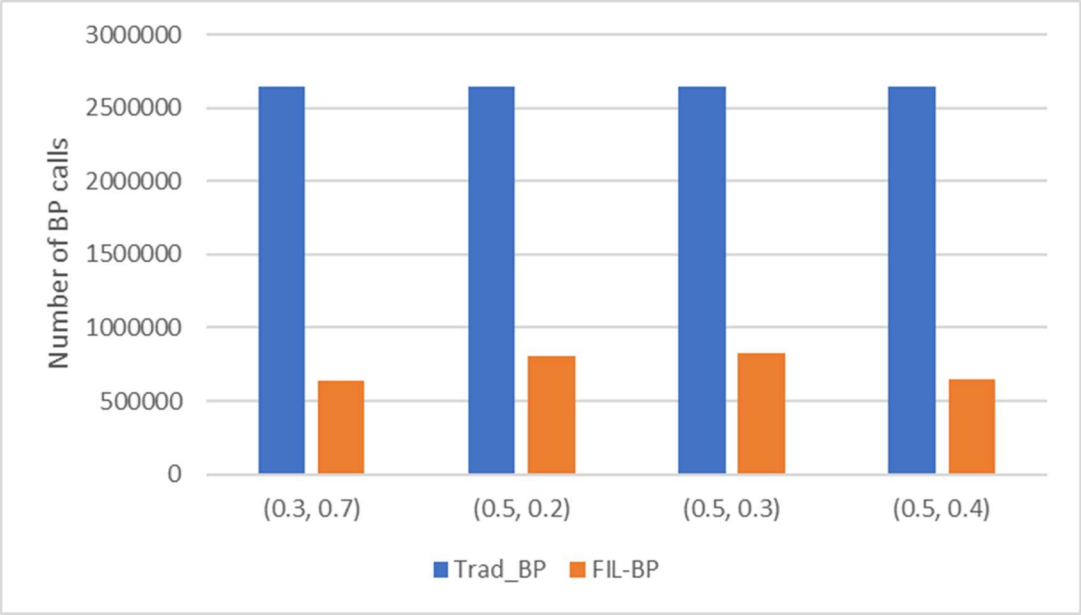


Figure 5.21: Number of BP function calls during different training sessions

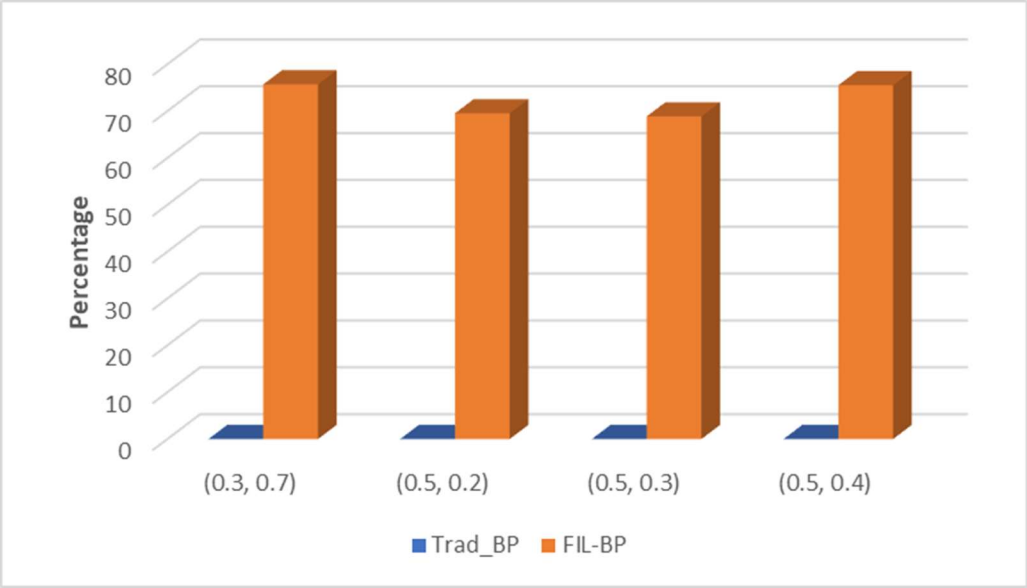


Figure 5.22: Percentage of BP function calls skipped during different training sessions

Figure 5.23 shows the system error difference between both networks. Notice the convergence level is included, but none of the networks under the current settings was able to achieve it.

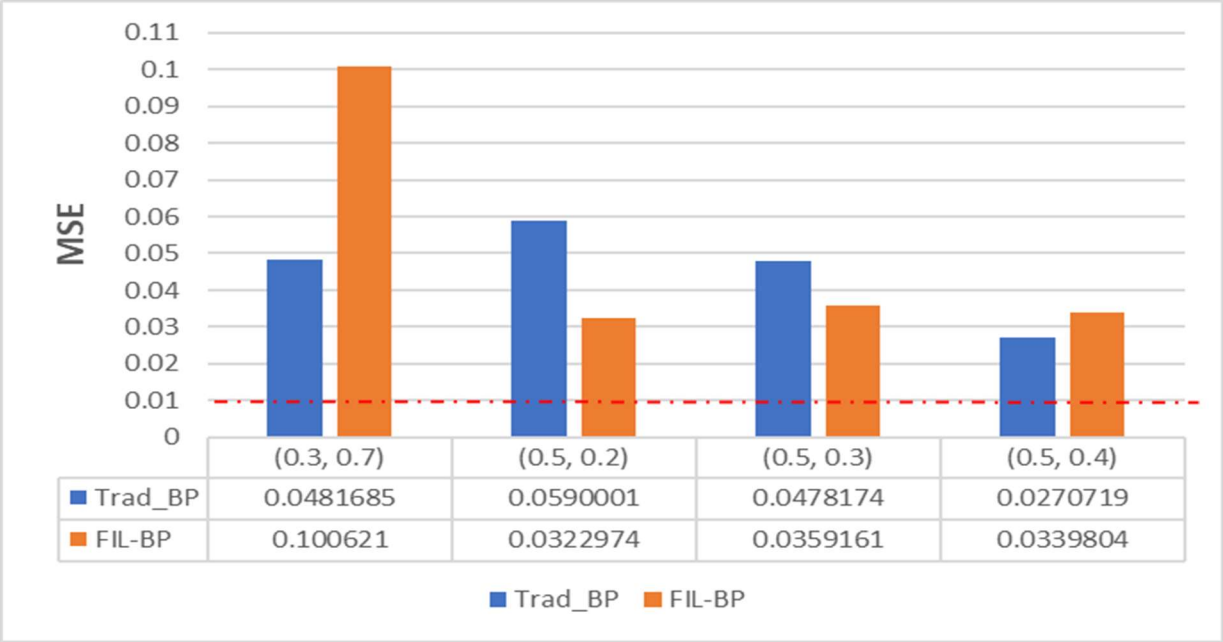


Figure 5.23: Final system error during different training sessions

Finally, Figure 5:24 shows that from all the training sessions, FIL-BP achieves the highest accuracy of all with a rate of 48.92%

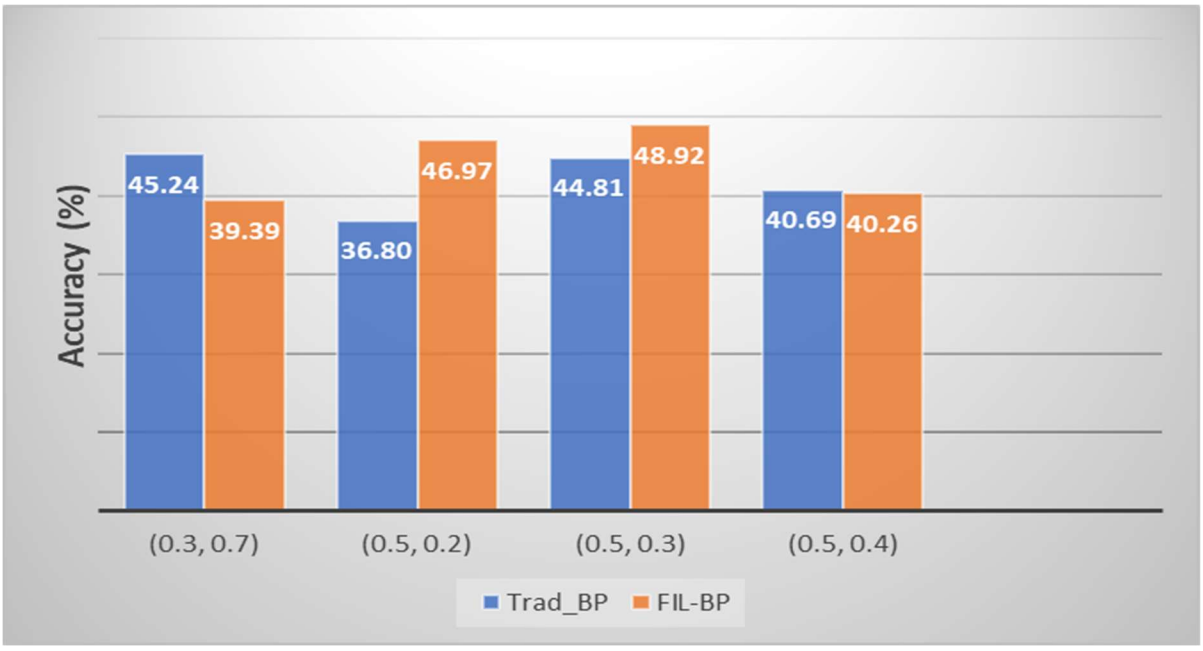


Figure 5.24: Classification after completing different training sessions

Lastly, in an attempt to summarize and highlight the satisfactory performance of FIL-BP over Trad-BP, the maximum, minimum and average values of each category across the 64 rows corresponding to each learning parameter combinations is listed on Table 5.6. Please note the complete table showing the comparison between both networks for each of the learning parameters used while training with the vowel database is found in Appendix D.

Table 5.6: Summary of training results for vowel database

		TRAINING *(Dataset size= 528)*			PERFORMANCE			EVALUATION *(Dataset size= 462)*	
		EPOCHS completed	FORWARD PHASE (# of function calls)	BACKWARD PHASE (# of BP function calls)	SAVINGS in BP (# of BP function calls skipped)	SAVINGS PERCENTAGE (% of BP skipped)	SYSTEM ERROR (MSE)	HITS	Classification Accuracy (%)
Trad-BP	*MAX*	5000	2,640,000	2,640,000	0	0	0.262952	209	45.24
	MIN	5000	2,640,000	2,640,000	0	0	0.027072	103	22.29
	AVG	5000	2,640,000	2,640,000	0	0	**0.075117**	172.25	37.28
FIL-BP	*MAX*	5000	2,640,000	2,536,024	1,998,475.00	75.70	0.240363	226	48.917749
	MIN	5000	2,640,000	641,525	103,976.00	3.94	0.032297	112	24.242424
	AVG	5000	2,640,000	1,128,580	**1,511,419.95**	**57.25**	**0.068177**	**176.06**	**38.11**

Notice on the table that FIL-BP is better on every category evaluating performance and generalization capability. On average, FIL-BP provides a 57% savings by the number of function calls that are skipped. This number is especially significant for the large databases that are typically part of problems solved by deep learning.

CHAPTER 6: CONCLUSION

The aim of this research is to investigate innovative strategies that can optimize the learning algorithms used in DLN with the purpose of speeding up the training phase in an effort to provide a solution to the challenging problem of reducing long training times that are typical in large AI applications.

This study shows that FIL-BP training model successfully reduces training execution times by inserting a two-phase intelligent fuzzy agent integrated into the backpropagation algorithm. During the first phase, the fuzzy agent serves as a decision support system which monitors the performance at the local level and decides when to allow the training procedure to continue and when to avoid updating the weights. In the second phase, the fuzzy agent uses performance metrics at the global level along with a rule base system to infer a conclusion that controls a skipping mechanism which successfully reduces the number of operations required to complete the training of a network while maintaining its classification accuracy.

This study shows that the network trained with the integrated fuzzy intelligent agent learning model successfully and substantially reduces the number of backpropagation operations by an average of at least 50% and provides up to 82% in savings to effectively reduce the execution time and accomplish the desired speedup. Remarkably, the extensive savings do not impact the ability for the network to reduce the system error, in fact, the network is able to achieve the same level of reduction as the traditional implementation and at times it outperforms it by reaching network convergence 600 epochs earlier which provide considerable extra savings by completing the training session in advance and avoiding all operations in the forward and backward training phases after that point, hence reducing the execution time and optimizing the training even further. Additionally, the generalization capability of FIL-BP model achieves the same high levels of classification accuracy, and in many instances, it outperforms the traditional implementation which is particularly noticeable on the most complex databased used.

The next sections present a concise review of the results, offers conclusions, and suggests recommendations for future work.

6.1 REVIEW OF RESULTS OBTAINED

A feedforward multi-layer network with binary sigmoid activation function trained with the traditional implementation of the backpropagation algorithm was created to serve as the baseline model identified as Trad-BP. The training scheme of the traditional backpropagation algorithm was modified to be assisted by fuzzy inference learning in the proposed model identified as FIL-BP.

In essence, FIL-BP model uses an intelligent agent that tracks the performance of the network as it processes the training data to produce a response. If the response to a particular pattern causes a *local* error considered acceptable to a tolerable degree, the fuzzy agent will advise the network to skip the backward phase of the training and focus on those patterns that cause larger errors, and hence need to complete the backpropagation process. By doing so, the network will effectively avoid those unnecessary updates to the weights which would not make a meaningful adjustment to the network's knowledge, nor contribute to the improvement of the classification capability. At the same time, the fuzzy agent considers the *global* performance of the network in order to control the skipping mechanism by diligently tracking the magnitude of the system error and the subtle changes in the error to infer an action that will modify the aforementioned tolerance when necessary.

In order to evaluate and compare their performance, both Trad-BP and FIL-BP models were trained with three different benchmark datasets that differ in size and complexity.

It should be noticed that while training for all three classification problems, FIL-BP was successfully able to skip numerous BP function calls causing substantial savings of up to 82%. This provided the desired training speedup effect while remarkably being able to maintain a system error and classification accuracy behaviors that effectively mirrored the baseline case.

These results are summarized by Table 6.1. Notice that, on average, the savings are at least 50% and can go up to 82% while the system error is maintained at the same comparable level. The classification accuracy of FIL-BP is actually increased in all categories for the large dataset.

Table 6.1: Performance comparison for Trad-BP and FIL-BP

		Trad- BP					FIL- BP				
		PERFORMANCE			EVALUATION		PERFORMANCE			EVALUATION	
		SAVINGS in # of BP function calls skipped	SAVINGS PERCENTAGE (% of BP skipped)	SYSTEM ERROR (MSE)	HITS	Classification Accuracy (%)	SAVINGS in # of BP function calls skipped	SAVINGS PERCENTAGE (% of BP skipped)	SYSTEM ERROR (MSE)	HITS	Classification Accuracy (%)
XOR	MAX	0	0	0.069264	4	100	2	0.110	0.069264	4	100
	MIN	0	0	0.009758	2	50	0	0	0.009758	2	50
	AVG	0	0	0.011162	3.94	98.44	0.03	0.00	0.011162	3.94	98.44
IRIS	MAX	0	0	0.355858	59	98.33	372,760	**82.836**	0.386947	59	98.33
	MIN	0	0	0.01	0	0	6,116	1.36	0.01	0	0
	AVG	0	0	0.054089	49.14	81.90	223,854	49.94	0.058380	47.45	79.09
VOWEL	MAX	0	0	0.262952	209	45.24	1,998,475.00	**75.70**	0.240363	**226**	**48.92**
	MIN	0	0	0.027072	103	22.29	103,976.00	3.94	0.032297	**112**	**24.24**
	AVG	0	0	0.075117	172.25	37.28	1,511,419.95	57.25	0.068177	**176.06**	**38.11**

6.2 CONCLUSION

Based on the observed results, several conclusions can be made. Overall, it can be stated that when FIL-BP was trained with three benchmark databases, it demonstrated an ability to provide great levels of savings which do not cause a significant impact on the minimization of the system error and does not hinder the classification accuracy. In fact, in the case of the vowel dataset, the generalization capability is greater for FIL-BP which seems to suggest that when the modified algorithm is trained with large, badly-behaved datasets, it also avoids the problem of

overtraining! This is a welcomed outcome because DL applications usually train with this type of big data.

By analyzing the empirical results across all different databases used, it can be concluded that FIL-BP can be recommended for general use independent of the type of problem. In fact, FIL-BP savings increases as the complexity, size, and challenge of the dataset increases.

For clarity of these comments, the following sections present the observation of each classification problem. Please refer to Table 6.1 for numerical values.

6.2.1 Comparing Performance While Training for the XOR Problem

It should be noted that the information presented for the XOR problem in Table 6.1 shows that the metrics under all the categories are the same for both Trad-BP and FIL-BP. There is only one opportunity of savings which is not significant. This means that on average, FIL-BP did not cause any speed up in training on this type of problem.

This result suggests that the proposed model doesn't find many opportunities for savings due to the small size of the database which contains only the core patterns necessary to achieve the expected learning.

6.2.2 Comparing Performance While Training for the IRIS Problem

The information presented for the IRIS problem in Table 6.1 shows that FIL-BP can provide substantially large savings of up to 82% on training, while, on average, the system error remains essentially the same when compared to Trad-BP (with a difference in error of 0.004). Similarly, there is a minimal classification accuracy difference of 2.81%. These minimal differences in system error magnitude and classification accuracy are a small price to pay when compared to the substantial savings that FIL-BP can provide by reducing the number of computations that are required during training. Based on the observed results, it could be argued that FIL-BP is recommended for use in medium size or well-behaved datasets, unless the aim of

the application strives to achieve the maximum classification accuracy possible, without regard to the time spent in training. By using FIL-BP the speedup provided by the average savings of at least 50% outweighs the 0.004 reduction error difference.

6.2.3 Comparing performance while training for the vowel problem

The information presented for the vowel problem in Table 6.1 shows that FIL-BP was able to provide savings of up to 75% with an average of 57% of BP function calls skipped. This means that on average FIL-BP allowed the system to skip more than half of the operations that would have been required under Trad-BP training.

Not only was FIL-BP able to offer big savings in computations made, it surprisingly was better in every category under the evaluation metrics. Notice that the number of hits while using the test data is greater in all cases, hence the classification accuracy is also greater than Trad-BP at the maximum, minimum and average comparisons.

These results suggest that for large and badly-behaved databases, FIL-BP not only speeds up the training, but also helps in the generalization. It could be argued that, by *not* spending time learning from training patterns that it already knows how to classify (within a tolerable degree) and hence avoiding unnecessary weight updates, the network also avoids overtraining which is an important factor that impacts generalization.

This argument can be supported by taking a closer look at the following case. When FIL-BP trained using the challenging vowel dataset, specifically under the learning parameters (α,μ)= (0.3, 0.7), it required the processing of 528 patterns that passed through the network for a duration of 5,000 epochs. This means that Trad-BP passed all the patterns forward and backward through the network causing a total of 2,640,000 updates to the weight matrix. Meanwhile, FIL-BP executed only 641,525 updates to the weigh matrix for a savings of 75%. By analyzing this result further, it is important to notice that under the same settings, at times FIL-BP model was skipping up to 89% of the whole dataset (skipping 470 of the 528 patterns) as shown in Figure 6.1.

This seems to suggest that by skipping so many patterns and reducing the training time, at the same time it avoided overfitting the network to the training data. This can be corroborated by the higher classification values.

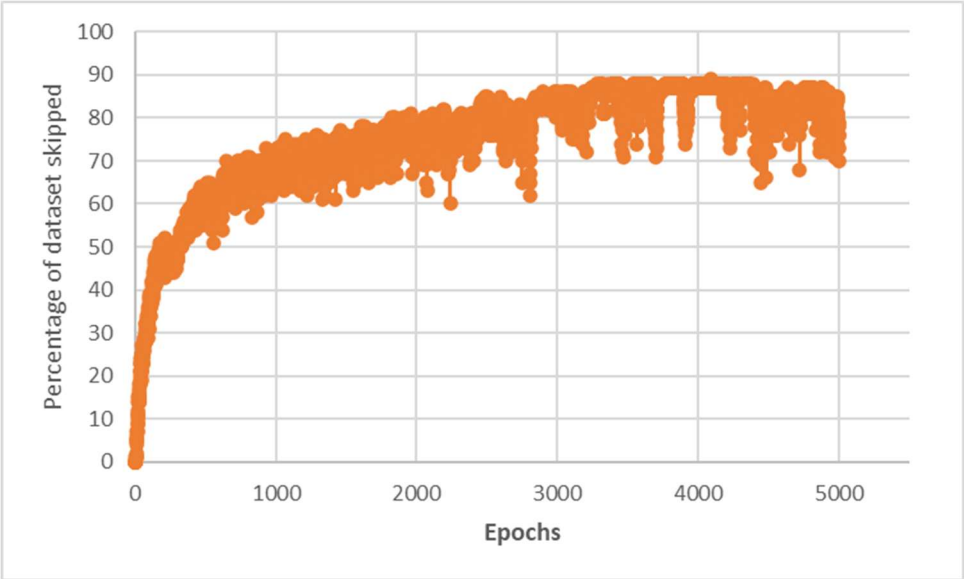


Figure 6.1: FIL-BP skips up to 89% of the data vectors in the dataset

6.3 Recommendations and Future Work

Based on the presented work and the obtained results, it would be interesting to continue the research investigating some of the following suggestions that could improve or enhance the proposed model.

First, since the model is already tracking the system error through the epochs and the change in error, it would be exciting to add more rules, and fine-tune the membership functions, for the fuzzy agent to use towards an exit strategy that would halt the training session when there is a trend that indicates the learning has stalled. For instance, if the error is small or the change in error is small and this trend continues for a certain number of epochs, at this point there is no more learning occurring, therefore, the system should stop training. This would allow a savings in the number of epochs required for training and would also skip all the operations in the forward phase.

Having this exit strategy would very likely avoid overtraining and also avoid the unnecessary modification of the weights that sometimes cause the final error to increase.

Second, using the FIL-BP model and shortening the training time would allow exploration of the effects of increasing the number of neurons or number of layers could have on problems such as the vowel classification where accuracy might improve by doing so. It also allows faster feedback on the effects of changing the number of maximum iterations or the use of optimizers (such as AdaGrad, RMS Prop or Adam) for the learning rate or momentum parameters.

Finally, the decision-making ability of the fuzzy agent could be improved by keeping track of each class in the dataset. By doing so, even if the pattern error is lower than the tolerance, the fuzzy agent can advise to continue with the backpropagation phase if the class performance is lower than desired.

REFERENCES

[Au98] Au Shu-Fai, "Letter Recognition Using a Fuzzy Logic Controlled Neural Network," M. S. Thesis, The University of Texas at El Paso, 1998.

[Ami18] Amiri Z., Hassanpour H., Khan M., Khan H., "Improving the Performance of Multilayer Backpropagation Neural Network with Adaptive Learning Rate", 2018 International Conference on Advances in Big Data, Computing and Data Communications Systems (icABCD), August, 2018

[Ara92] Arabshahi P., Choi J., Marks R., Caudell T., "Fuzzy Control of Backpropagation", IEEE International Conference on Fuzzy Systems, Proceedings 967-972, 1992

[Ben12] Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In Neural Networks: Tricks of the Trade. (pp. 437-478). Berlin: Springer.

[Chu01] Chudler, H. Eric, "A computer in your head?" *Odyssey magazine, Cobblestone Publishing,* 10:6-7, 2001.

[Cir12] Ciresan, D., Meier U., Schmidhuber J., ""Multi-column deep neural networks for image classification," in Computer Vision and Pattern Recognition (CVPR), no. February, pp. 3642-3649, 2012.

[Dua19] Dua, D., "UCI Machine Learning Repository," (University of California, School of Information and Computer Science) http://archive.ics.uci.edu/ml, January 1, 2019.

[Fau94] Fausset, Laurene, "Fundamentals of Neural Networks: Architectures, Algorithms, and Applications," *Prentice Hall* , pp1-96,289-324, 1994.

[Fal21] Falk, A., "Deep Learning vs Machine Learning: What's the Difference," IEEE Computer Society Tech News, https://lawtomated.com/a-i-technical-machine-vs-deep-learning/, June 15, 2021.

[Fis36] Fisher, R., "The use of multiple measurements in taxonomic problems," Annual Eugenics, 179-188, 1936.

[Fre01] Freudenrich, C. Craig, "How your brain works," http://science.howstuffworks.com/brain.htm/, 2001.

[Hai21] Human-centered Artificial Intelligence, "AI Index Report 2021," https://aiindex.stanford.edu/wp-content/uploads/2021/03/2021-AI-Index-Report_Master.pdf

[He15] He, K., Zhang X., Ren S., Sun J., "Delving deep into rectifers: Surpassing human-level performance on imagenet classification," Proceedings of the IEEE International Conference on Computer Vision, vol. 2015 Inter, pp. 1026-1034, 2015.

[Ho19] Ho, K., "MIT Technology Review," TechnologyReview.com: https://www.technologyreview.com/2019/06/06/239031/training-a-single-ai-model-can-emit-as-much-carbon-as-five-cars-in-their-lifetimes/, June 6, 2019

[Gax12] Gaxiola F., Melin P., Valdez F., "Backpropagation method with type-2 Fuzzy Weight Adjustment for Neural Network Learning", Proceedings of the International Joint Conference on Neural Networks (IJCNN), 2012

[Jan97] Jang J. -S. R., Sun C.-T., Mizutani E., "Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence," Prentice Hall, pp.197-327, 1997.

[Jou17] Jouppi N.P., Borchers A., Boyle R., et al, "In-Datacenter Performance Analysis of a Tensor Processing Unit," ACM SIGARCH, Computer Architecture News, vol. 45, no. 2, pp. 1-12, 2017.

[Kar04] Karakose M., Akin E., "Type-2 fuzzy activation function for multilayer feedforward neural networks" , IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No 04CH37583), October, 2004

[Kar16] Karn, U., "An Intuitive Explanation of Convolutional Neural Networks," https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/, August 11, 2016.

[Kau20] Kaur, J., "Automatic log analysis using deep learning and AI," https://www.xenonstack.com/blog/data-science/log-analytics-deep-machine-learning-ai/, August 24, 2020

[Klu22] Klubnikin, A., "How much does artificial intelligence cost?," https://itrexgroup.com/blog/how-much-does-artificial-intelligence-cost/, August 29, 2022

[Kol19] Kolbusz J., Lysenko O., Rozycki P., Wilamowski B., "Error Back Propagation Algorithm with Adaptive Learning Rate", International Conference on Information and Digital Technologies (IDT, June 2019

[Kri17] Krizhevsky, A., Sutskever, I., & Hinton, G. E., "Imagenet classification with deep convolutional neural networks," Communications of the ACM, (pp. 84-90), 2017.

[Lab21] Lab, S. V., "Imagenet," https://image-net.org/, March 11, 2021

[Lar19] Lardinois, F., "Google's newest Cloud TPU Pods feature over 1,000 TPUs," https://techcrunch.com/2019/05/07/googles-newest-cloud-tpu-pods-feature-over-1000-tpus/, Tech Crunch, May 7, 2019.

[Lin15] Ling A., Capalija D., Chiu G., "Accelerating Deep Learning with the OpenCL Platform and Intel Stratix 10 FPGAs," tech. rep., Intel, 2015.

[Lip94] Lippe W. M., Feuring Th., Tenhagen A., "A Fuzzy-controlled Delta-bar-delta Learning rule", Proceedings of IEEE International Conference on Neural Networks (ICNN;94), July 1994

[Mcc43] McCulloch W. S. And Pitts W., "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, 5:115-133, 1943.

[Mlp20] MLPerf Training V0.7 Results 2020, https://mlcommons.org/en/

[Nok19] Nokland A., Eidnes L., "Training Neural Networks with Local Error Signals", International Conference of Machine Learning, 2019

[Ovt15] Ovtcharov K., Ruwase O., Kim J., Fowers J., Strauss K., Chung E.S., "Accelerating Deep Convolutional Neural Networks Using Specialized Hardware," Microsoft Research Whitepaper, pp. 3-6, 2015.

[Ras12] Rashidy H., Azar K., "Reduction of Neural Network Training Time Using an Adaptive Fuzzy Approach in Real Time Applications" , International Journal of Information and Electronics Engineering, Vol. 2, No. 3, May 2012

[Rud76] Rudin, W., "Principles of Mathematical Analysis," McGraw-Hill, 1976.

[Rus20] Rustum, R., Kurichiyanil, Forrest, S., & Sommariva, C., "Sustainability Ranking of Desalination Plants using Mamdani Fuzzy Logic Inference System," Sustainability, 12(2) 631, 2020.

[Sch02] Schmutter, P., "A Taxonomy for Artificial and Computational Intelligence," http://www.ooop.org/publications/thesis/node18.html, January 1, 2002.

[Sch19] Schuchmann, S., "History of the first AI Winter," https://towardsdatascience.com/history-of-the-first-ai-winter-6f8c2186f80b, Towards Data Science, May 12, 2019.

[Tch17] Tch, A., "The mostly complete chart of Neural Networks, explained," https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464, Towards Data Science, August 4, 2017.

[Wig21] Wiggers, K., "AI weekly: AI model training costs on the rise, highlighting need for new solutions," https://venturebeat.com/ai/ai-weekly-ai-model-training-costs-on-the-rise-highlighting-need-for-new-solutions/, The Machine, October 15, 2021

[Wil19] Willems, K., "Keras Tutorial: Deep Learning in Python," https://www.datacamp.com/community/tutorials/deep-learning-python, Datacamp, December 1, 2019.

APPENDIX A: Program Code:

Backpropagation Algorithm with Trad-BP and FIL-BP modes

```
/*************************************************************
 *     Optimized learning using fuzzy-inference-assisted
 *          algorithms for Deep Learning
 *
 *                       Miroslava Barúa
 *
 *          A dual mode feedforward multi-layer neural network
 *          with binary sigmoid activation function trained
 *          with choice of traditional Backpropagation (BP) algorithm
 *          or optimized BP training with Fuzzy intelligent agent.
 *
 *       Basic Version
 *
 *               Define BP net architecture
 *               Initialize weight matrix w/small random #'s
 *                Initialize fuzzy inference system
 *               Obtain training data from specified file
 *               Augment input vector with bias neuron
 *               Train net with backpropagation algorithm
 *               Do Forward phase
 *               Use Fuzzy Inference System to optimize backward phase
 *               Control BP skip conditions
 *               Process test data obtained from file
 *               Evaluate performance
 *
 ************************************************************/

/*************************************************************
 *    Portions of code for traditional BP mode
 *    based on code by Dr.  Patricia Nava
 *
 *************************************************************/

/*************************************************************
 *
 *    File Name:  FIL-BP.c
 *
 *************************************************************/


/*************************************************************
 *    File contains the following functions:
 *
 *    back_prop(i):
 *    get_goals():
*   init_inputs():
```

```
*   init_weights():
    *   main():
    *   process():
    *   propagate(i):
    *   rnd_num():
    *   train():
    *   init_fuzzy_system()
    *   get_system_inputs(input1, input2)
    *   fuzzification()
    *   rule_evaluation()
    *   defuzzification()
    *
    ************************************************************/

    /*---------------- INCLUDE FILES -------------------------*/

#include "stdio.h"
#include "math.h"
#include "stdlib.h"
#include "float.h"
#include <string.h>

#define IN_UNTS    2          /* no. of input neurons */
#define HDN_UNTS   2          /* no. of hidden layer neurons */
#define OUT_UNTS   1          /* no. of output neurons */
#define FEED_IN    4          /* max no. of inputs to a neuron */
#define N_THRES    1.0        /* threshold for neurons if not in file */
#define MX_UNTS    4          /* max no. of neurons in any layer */
#define MX_PATTS   4          /* max no. of input samples */
#define MX_INPTS   16         /* max no. of input neurons */
#define MX_OUTPTS  3          /* max no. of output neurons */
#define NET_THRESHOLD 1.0   /*neurons threshold value */


#define max(a,b)   (a<b ? b : a)     /* For Fuzzy agent*/
#define min(a,b)   (a>b ? b : a)     /* rule evaluation */
#define MAXNAME 10
#define UPPER_LIMIT 1             /*limit - degree of membership */


/*---------------- TYPEDEFS AND STRUCTS BP -------------------*/

typedef struct {
    float input[FEED_IN + 1];
    float weight[FEED_IN + 1];
```

```
  float delta_weight[FEED_IN + 1];
    float delta;
    float output;
} neuron;                                   /*define neuron's structure*/

float input[MX_PATTS][MX_INPTS + 1];    /*declare matrices by architecture*/
float goals[MX_PATTS][MX_OUTPTS];
float known_output[MX_PATTS][MX_OUTPTS];
float patt_err[MX_PATTS][MX_OUTPTS];
float tot_patt_err[MX_PATTS];

neuron hid_layer[HDN_UNTS + 1];
neuron out_layer[OUT_UNTS];

float sys_err;                              /* system error */
int num_samples, num_goals, num_misses = 0;
int max_iterations = 5000, min_err_iter;       /* Set training session maximum duration */
int miss_flag = 0;
float learning_rate = 0.7, momentum = 0.8;          /* learning parameters*/
float max_tot_err = 0.01, max_indv_err = 0.001;     /* system error tolerance*/
float min_err_td = 500.0;
long randseed = 568731L;                    /* for random number*/
char rpt_name[20];


/*---------------- TYPEDEFS AND STRUCTS FOR FUZZY AGENT -------------------*/

struct io_type* System_Inputs;
struct io_type* System_Output;
struct rule_type* Rule_Base;       /* rules in Rule base*/

struct io_type {
   char name[MAXNAME];            /* FIS system input and output structures  */
   float value;
   struct mf_type* membership_functions;
   struct io_type* next;
};
struct mf_type {                  /* Membership functions structure*/
   char name[MAXNAME];
   float value;           /* degree of membership */
   float point1;
   float point2;
   float slope1;
   float slope2;
   struct mf_type* next;
};
```

```
struct rule_type {
        struct rule_element_type* if_side;      /* antecedents */
        struct rule_element_type* then_side;    /* consequent */
        struct rule_type* next;
      };
      struct rule_element_type {
        float* value;
        struct rule_element_type* next;     /* next element in rule */
      };

      float my_patt_err[MX_PATTS];          /* total local patterrn error ej */
      float epoch_sys_err;          /* epoch error*/
      float outFIS;
      int skip = 0;
      int iter;
      int skip_BPcnt = 0;
      int yes_bp_cnt = 0;
      int my_skip_cnt = 0;
      float pat_err_tresh = 0.0;
      /* Set up pointers */
      FILE* rpt_ptr;
      FILE* wt_ptr;
      FILE* wtfin_ptr;

      /*****************************************************************
       *
       *  FUNCTION NAME: rnd_num()
       *
       *  PARAMETERS: none
       *
       *  RETURN VALUE: random floating point number
       *
      ******************************************************************/

        /* Get small  random number for neuron's weight matrix */
      float rnd_num()
      {
        int num;
        randseed = 15625L * randseed + 22221L;
        num = (randseed >> 16) & 0x7FFF;
        return(num / pow(2.0, 15.0) - 0.5);
      }/* end of rnd_num() function */

      /*****************************************************************
       *
*  FUNCTION NAME: init_weights()
```

```
 *
 *  PARAMETERS: none
 *
 *  RETURN VALUE: void
 *
 *  DESCRIPTION:  user initializes  neuron's weights with small
 *                        random values --> calling generator function.
 *
 ************************************************************/
void init_weights()
{
   int j, k;
   float rnd_num();
   /*  set up hidden layer threshold neuron */
   hid_layer[HDN_UNTS].output = N_THRES;

   for (j = 0; j < HDN_UNTS; j++) {            /* hidden neuron # */
      for (k = 0; k < IN_UNTS + 1; k++) {              /*input to jth neuron */
         hid_layer[j].weight[k] = rnd_num();
      } /* endloop (# of weights for each unit) */
   }/* endloop (# of hidden units) */

   for (j = 0; j < OUT_UNTS; j++) {            /*ouput neuron # */
      for (k = 0; k < HDN_UNTS + 1; k++) {          /*input to jth neuron */
         out_layer[j].weight[k] = rnd_num();
      }/* endloop (# weights for each unit) */
   }/* endloop (# of output units) */

   /*----------- Progress information ---------*/
     /* Write initial weight matrix onto a file */
    /* Loop for the hidden layer neuron's weights */
   fprintf(wt_ptr, "\n\n\n----------------------------------------------");
   fprintf(wt_ptr, "\nweights are:\n");
   fprintf(wt_ptr, "hidden_layer \n");

   for (j = 0; j < HDN_UNTS; j++) {        /* write hidden layer's weights  */
      fprintf(wt_ptr, " \n");
      for (k = 0; k < IN_UNTS + 1; k++) {
         fprintf(wt_ptr, " %f \t\t", hid_layer[j].weight[k]);
      }/*for all inputs*/
   }/*for all hidden units*/

 /* Loop for the output layer neuron's weights */
   fprintf(wt_ptr, "\noutput_layer \n");
   for (j = 0; j < OUT_UNTS; j++) {        /* write output layer's weights  */
```

```
fprintf(wt_ptr, " \n");
            for (k = 0; k < HDN_UNTS + 1; k++) {
               fprintf(wt_ptr, " %f \t\t", out_layer[j].weight[k]);
            }/*for all hidden units*/
         }/*for all output units*/
      }/* end init_weight function */


      /*****************************************************************
       *
       *  FUNCTION NAME: init_inputs()
       *
       *  PARAMETERS: none
       *
       *  RETURN VALUE: void
       *
       *  DESCRIPTION:  Reads TEST patterns to process from a file to eval NN
       *
       *****************************************************************/
      void init_inputs()
      {
         int j, k;
         FILE* f_ptr;
                  /*error message when file unavailable */
         if ((f_ptr = fopen("Test.txt", "r")) == NULL) {
            fprintf(rpt_ptr, "\nThe input file could");
            fprintf(rpt_ptr, " not be opened. Terminating program.");
            exit(10);
         }/* endif */
            /*Testing file scan */
         if (fscanf(f_ptr, "%d", &num_samples) != EOF)
            fprintf(rpt_ptr, "\n\n Opening test data of %d patterns\n\n ", num_samples);
         else
            printf("init_inputs() error ~ bad num_samples \n");
         for (j = 0; j < num_samples; j++) {
            for (k = 0; k < IN_UNTS; k++) {
               if (fscanf(f_ptr, "%f", &input[j][k]) != EOF)
                  printf("....\n");
               else {
                  printf("init_inputs() error ~ in input units or reached EOF\n");
                  return(0);
               }
            }/* endloop (# of input units) */
              /* set up threshold "neuron" for input layer */
            input[j][IN_UNTS] = 1.0;
              /* ---------- this section for eval. purposes ---*/
```

```
for (k = 0; k < OUT_UNTS; k++) {
          if (fscanf(f_ptr, "%f", &known_output[j][k]) != EOF)
             printf("....");
          else
             printf("Error in target values \n");
       }/* endloop (# of target output units) */
    }/* endloop (# of data sets) */
    fclose(f_ptr);
} /* end of init_inputs() function */


/*********************************************************************
 *
 *  FUNCTION NAME: init_fuzzy_system()
 *
 *  PARAMETERS: none
 *
 *  RETURN VALUE: void
 *
 *  DESCRIPTION:  Reads files for antecedents, consequence, rule base
 *                init corresponding structures, determine slopes
 *
 *********************************************************************/

init_fuzzy_system()
{
    float a, b, c, d, x;
    char buff[10], buff1[4], buff2[4];
    FILE* fp;
    struct io_type* outptr;     /*set pointers*/
    struct mf_type* top_mf;
    struct mf_type* mfptr;
    struct io_type* ioptr;
    struct rule_type* ruleptr;
    struct rule_element_type* ifptr;
    struct rule_element_type* thenptr;
    ioptr = NULL;
    ruleptr = NULL;
    ifptr = NULL;
    thenptr = NULL;
    top_mf = NULL;

    /*~~~ INITIALIZE STRUCTURES */
    if ((fp = fopen("Antecedent1.txt", "r")) == NULL)   /* First antecedent */
    {
       printf("ERROR- Unable to open file named Antecedent1.txt.\n");
```

```
exit(0);
   }
   else {
       printf("Succes open file\n");
   }
         /* set pointer*/
   ioptr = (struct io_type*)calloc(1, sizeof(struct io_type));
   System_Inputs = ioptr;
   x = fscanf(fp, "%s", buff);
   sprintf(ioptr->name, "%s", buff);
   mfptr = NULL;
   while ((x = fscanf(fp, "%s %f %f %f %f", buff, &a, &b, &c, &d)) != EOF)
   {
      if (mfptr == NULL)
      {
         mfptr = (struct mf_type*)calloc(1, sizeof(struct mf_type));
         top_mf = mfptr;
         ioptr->membership_functions = mfptr;
      }
      else
      {
         for (mfptr = top_mf; mfptr->next; mfptr = mfptr->next);
         mfptr->next = (struct mf_type*)calloc(1, sizeof(struct mf_type));
         mfptr = mfptr->next;
      }
      sprintf(mfptr->name, "%s", buff);
      mfptr->point1 = a;
      mfptr->point2 = d;

      if ((b - a) == 0) {
         mfptr->slope1 = (UPPER_LIMIT / 1);
      }
      else if (b - a > 0) {
         mfptr->slope1 = (UPPER_LIMIT / (b - a));
      }
      else
      {
         printf("Error (b-a)<0 element %s.\n", buff);
         exit(1);
      }
      if ((d - c) == 0) {
         mfptr->slope2 = (UPPER_LIMIT / 1);
      }
      else if (d - c > 0) {
         mfptr->slope2 = UPPER_LIMIT / (d - c);
      }
```

```
else {
                printf("Error in (d-c)<0  element %s.\n", buff);
                exit(1);
            }
        }
        fclose(fp);

        /*~~~ INITIALIZE STRUCTURES */

        if ((fp = fopen("Antecedent2.txt", "r")) == NULL)
        {
            printf("ERROR- Unable to open data file Antecedent2.txt.\n");
            exit(0);
        }
        else
        {
            printf("\n Success open file\n");
        }       /* set pointer*/
        ioptr->next = (struct io_type*)calloc(1, sizeof(struct io_type));
        ioptr = ioptr->next;
        x = fscanf(fp, "%s", buff);
        sprintf(ioptr->name, "%s", buff);
        mfptr = NULL;
        while ((x = fscanf(fp, "%s %f %f %f %f", buff, &a, &b, &c, &d)) != EOF)
        {
            if (mfptr == NULL)
            {
                mfptr = (struct mf_type*)calloc(1, sizeof(struct mf_type));
                top_mf = mfptr;
                ioptr->membership_functions = mfptr;
            }
            else
            {
                for (mfptr = top_mf; mfptr->next; mfptr = mfptr->next);
                mfptr->next = (struct mf_type*)calloc(1, sizeof(struct mf_type));
                mfptr = mfptr->next;
            }
            sprintf(mfptr->name, "%s", buff);
            mfptr->point1 = a;
            mfptr->point2 = d;

            /* calculate slopes and store*/
            if ((b - a) == 0) {
                mfptr->slope1 = (UPPER_LIMIT / 1);
            }
            else if (b - a > 0) {
```

```
mfptr->slope1 = (UPPER_LIMIT / (b - a));
                }
                else
                {
                   printf("Error (b-a)<0 element %s.\n", buff);
                   exit(1);
                }

                if ((d - c) == 0) {
                   mfptr->slope2 = (UPPER_LIMIT / 1);
                }
                else if (d - c > 0) {
                   mfptr->slope2 = UPPER_LIMIT / (d - c);
                }
                else {
                   printf("Error in (d-c)<0 element %s.\n", buff);
                   exit(1);
                }
             }
             fclose(fp);

             /****** INITIALIZE STRUCTURES */
             if ((fp = fopen("Conseq_out.txt", "r")) == NULL)
             {
                printf("ERROR- Unable to open data file Conseq_out.txt.\n");
                exit(0);
             }
             else {
                printf("\n Opened successfully\n");
             }
                /* set pointer */
             ioptr = (struct io_type*)calloc(1, sizeof(struct io_type));
             System_Output = ioptr;
             x = fscanf(fp, "%s", buff);
             sprintf(ioptr->name, "%s", buff);
             mfptr = NULL;
             while ((x = fscanf(fp, "%s %f %f %f %f", buff, &a, &b, &c, &d)) != EOF)
             {
                if (mfptr == NULL)
                {
                   mfptr = (struct mf_type*)calloc(1, sizeof(struct mf_type));
                   top_mf = mfptr;
                   ioptr->membership_functions = mfptr;
                }
                else
                {
```

```
for (mfptr = top_mf; mfptr->next; mfptr = mfptr->next);
                mfptr->next = (struct mf_type*)calloc(1, sizeof(struct mf_type));
                mfptr = mfptr->next;
            }
            sprintf(mfptr->name, "%s", buff);
            mfptr->point1 = a;
            mfptr->point2 = d;

            if ((b - a) == 0) {
                mfptr->slope1 = (UPPER_LIMIT / 1);
            }
            else if (b - a > 0) {
                mfptr->slope1 = (UPPER_LIMIT / (b - a));
            }
            else
            {
                printf("Error (b-a)<0 in element %s.\n", buff);
                exit(1);
            }
            if ((d - c) == 0) {
                mfptr->slope2 = (UPPER_LIMIT / 1);
            }
            else if (d - c > 0) {
                mfptr->slope2 = UPPER_LIMIT / (d - c);
            }
            else {
                printf("Error in (d-c)<0 element %s.\n", buff);
                exit(1);
            }
        }
        fclose(fp);

         /***INITIALIZE STRUCTURES */
        ioptr = NULL;
        outptr = NULL;
        if ((fp = fopen("rulebase.txt", "r")) == NULL)        {
            printf("ERROR- Unable to open RULES file successully.\n");
            exit(0);
        }
        else {
            printf("\n Success open file\n\n");
        }
        ruleptr = (struct rule_type*)calloc(1, sizeof(struct rule_type));
        if (ioptr == NULL)Rule_Base = ruleptr;
        while ((x = fscanf(fp, "%s %s %s", buff, buff1, buff2)) != EOF)
        {
```

```
ioptr = System_Inputs;
            for (mfptr = ioptr->membership_functions; mfptr != NULL; mfptr = mfptr->next)
            {
               if ((strcmp(mfptr->name, buff)) == 0)       /* first*/
               {
                  ifptr = (struct rule_element_type*)
                     calloc(1, sizeof(struct rule_element_type));
                  ruleptr->if_side = ifptr;
                  ifptr->value = &mfptr->value;
                  ifptr->next = (struct rule_element_type*)
                     calloc(1, sizeof(struct rule_element_type));
                  ifptr = ifptr->next;
                  break;
               }
               else {
                  printf("...");
               }
            }
            ioptr = ioptr->next;                /* second  */
            for (mfptr = ioptr->membership_functions; mfptr != NULL; mfptr = mfptr->next)
            {
               if ((strcmp(mfptr->name, buff1)) == 0)
               {  ifptr->value = &mfptr->value;
                  break;
               }
               else {
                  printf("...\n");
               }
            }
            if (outptr == NULL)outptr = System_Output;  /* output  */
            for (mfptr = outptr->membership_functions; mfptr != NULL; mfptr = mfptr->next)
            {   if ((strcmp(mfptr->name, buff2)) == 0)
               {  thenptr = (struct rule_element_type*)
                     calloc(1, sizeof(struct rule_element_type));
                  ruleptr->then_side = thenptr;
                  thenptr->value = &mfptr->value;
                  break;
               }
               else {
                  printf(" ...\n");
               }
            }
            ruleptr->next = (struct rule_type*)calloc(1, sizeof(struct rule_type));
            ruleptr = ruleptr->next;
         }            /* FINISHED FILES */
   fclose(fp);
```

```
}      /* END INITIALIZE FUZZY SYSTEM */

/*************************************************************
 *
 *  FUNCTION NAME: propagate(i)
 *
 *  PARAMETERS: i -- integer pattern number
 *
 *  RETURN VALUE: void
 *
 *  DESCRIPTION:  FORWARD PHASE, propagates pattern #i through the net

 *************************************************************/
void propagate(i)                  /*feedforward pass begins */
int i;
{
   int j, k;
   float sum, fnet;
   /* ---------- initialize & calc. hidden units' responses --- */
   for (j = 0; j < HDN_UNTS; j++) {
      for (k = 0; k < IN_UNTS + 1; k++) {
         hid_layer[j].input[k] = input[i][k];
      }/* endloop (initializing inputs for this unit with scanned values) */
   }/* endloop (initializing inputs for hidden units with scanned values) */

   for (j = 0; j < HDN_UNTS; j++) {
      sum = 0.0;                           /*weighted sum initialized*/
      for (k = 0; k < IN_UNTS + 1; k++) {
         sum += hid_layer[j].weight[k] * hid_layer[j].input[k];
      }/* endloop (summing weighted inputs) */

      fnet = -(sum) / NET_THRESHOLD;    /*hidden neuron's activation*/
      hid_layer[j].output = 1.0 / (1.0 + exp(fnet));      /*function response*/
   }/* endloop (calculations of hidden layer) */

   /* ---------- initialize & calc. output units' responses --- */
   for (j = 0; j < OUT_UNTS; j++) {
      for (k = 0; k < HDN_UNTS + 1; k++) {
         out_layer[j].input[k] = hid_layer[k].output;
      }/* endloop (initializing inputs for this unit with scanned values) */
   }/* endloop (initializing inputs for hidden units with scanned values) */

   for (j = 0; j < OUT_UNTS; j++) {
      sum = 0.0;                           /*weighted sum initialized*/
      for (k = 0; k < HDN_UNTS + 1; k++) {
         sum += out_layer[j].weight[k] * out_layer[j].input[k];
```

```
}/* endloop (summing weighted inputs) */

            fnet = -(sum) / NET_THRESHOLD;     /*output neuron's activation*/
            out_layer[j].output = 1.0 / (1.0 + exp(fnet));      /*function response*/
         }/* endloop (calculations of output layer) */
      }/* end of FWD propagate function */

      /*************************************************************
       *
       *  FUNCTION NAME: get_goals()
       *
       *  PARAMETERS: none
       *
       *  RETURN VALUE: void
       *
       *  DESCRIPTION:  reads training patterns from file
       *              # of patterns is the first piece of data in  the file
       *              followed by input patterns
       *************************************************************/
      void get_goals()
      {
         int j, k;
         FILE* f_ptr;

         /*---------------- OPEN TRAINING DATA FILE ------------------*/
         if ((f_ptr = fopen("TRAIN_file.txt", "r")) == NULL) {
            fprintf(rpt_ptr, "\nThe input file for training could");
            fprintf(rpt_ptr, " not be opened. Terminating program.");
            exit(11);
         }/* endif */
         fscanf(f_ptr, "%d", &num_goals);
         printf("Successfully Opened training file\n");
         for (j = 0; j < num_goals; j++) {
            for (k = 0; k < IN_UNTS; k++) {
               fscanf(f_ptr, "%f", &input[j][k]);
            }/* endloop (# of input units) */
           /* set up threshold "neuron" */
            input[j][IN_UNTS] = 1.0;
            for (k = 0; k < OUT_UNTS; k++) {
               fscanf(f_ptr, "%f", &goals[j][k]);
            }/* endloop (# of target output units) */
         }/* endloop (# of training data sets) */
         fclose(f_ptr);
      }/* end of get_goals() function */

      /*************************************************************
```

```
*
 *  FUNCTION NAME: set_params()
 *
 *  PARAMETERS: none
 *
 *  RETURN VALUE: void
 *
 *  DESCRIPTION:  allows user to set parameters and report,
 *                init delta weights (initialize to zero)
 ************************************************************/
void set_params()
{    int j, k;
   char choice, fname[20];
   /*indicate status on report file */
   fprintf(rpt_ptr, "\n\n\n----------------------------------------------");
   fprintf(rpt_ptr, "\nParameters are:\nlearning_rate = %f", learning_rate);
   fprintf(rpt_ptr, "\tmomentum = %f", momentum);
   /* initialize delta_weights BEFORE training */
   for (j = 0; j < HDN_UNTS; j++) {
      for (k = 0; k < IN_UNTS + 1; k++) {
         hid_layer[j].delta_weight[k] = 0.0;
      }/* weights on all inputs */
   }/* all of the hidden units */
   for (j = 0; j < OUT_UNTS; j++) {
      for (k = 0; k < HDN_UNTS + 1; k++) {
         out_layer[j].delta_weight[k] = 0.0;
      }/* weights on all inputs */
   }/* all of the output units */
}/* end set_params function */

/************************************************************
 *
 *  FUNCTION NAME: fuzzytutor(i)
 *
 *  PARAMETERS: i - integer pattern number

 *
 *  RETURN VALUE: 0 or 1
 *
 *  DESCRIPTION:  will calculate Pattern [i] FWD pass total error
 *                (accross ALL output neurons) and decide if
 *                 BP function call or not
 *
 ************************************************************/

int fuzzytutor(i)
```

```
int i;
        {
           int j, k;
           float sum;
           my_patt_err[i] = 0.0;  /*pattern total error*/
           int my_grade = 0.0;
           for (j = 0; j < OUT_UNTS; j++) {   /*compare target with output*/
              patt_err[i][j] = (goals[i][j] - out_layer[j].output)
                 * (goals[i][j] - out_layer[j].output);
              my_patt_err[i] += patt_err[i][j];            /*sum errors*/
           }/* for all output units */
           if ((my_patt_err[i] > pat_err_tresh)) {
              yes_bp_cnt++;                        /*continue */
              my_grade = 1;
           } /*for all patterns sent to do BP */
           else {
              my_skip_cnt++;
              my_grade = 0;
           };
           return (my_grade);      /* return decision*/
        }/* end of fuzzytutor() function */

        /*************************************************************
         *
         *  FUNCTION NAME: get_system_inputs(input1, input2)
         *
         *  PARAMETERS: input1, input2
         *
         *  RETURN VALUE: void
         *
         *  DESCRIPTION:  Get system inputs intercepted
        *              from BP into structure
         *             to be processed by fuzzy agent
         * ************************************************************/
        float get_system_inputs(input1, input2)
        float input1, input2;
        {  struct io_type* ioptr;
        ioptr = System_Inputs;
        ioptr->value = input1;      /* first  value*/
        ioptr = ioptr->next;
        ioptr->value = input2;      /*second  value*/
        }    /*end of get_system_inputs*/

        /*************************************************************
         *
         *  FUNCTION NAME: fuzzification ()
```

```
*
 *  PARAMETERS: none
 *
 *  RETURN VALUE: void
 *
 *  DESCRIPTION:  Uses the slopes, structure and input values
 *                to fuzzify each input value into corresponding
 *               DEGREE OF MEMBERSHIP for each set!
 *                (values correspond to antecedent in rules)
 *
 ************************************************************/

fuzzification()
{
   struct io_type* si;        /*system input pointers*/
   struct mf_type* mf;
                //FUZZIFICATION begins
   for (si = System_Inputs; si != NULL; si = si->next)
      for (mf = si->membership_functions; mf != NULL; mf = mf->next)
      {
         compute_degree_of_membership(mf, si->value);
      }
}   /*end of fuzzification*/

/************************************************************
 *
 *  FUNCTION NAME: rule_evaluation ()
 *
 *  PARAMETERS: none
 *
 *  RETURN VALUE: void
 *
 *  DESCRIPTION:  When rule is evaluated, use min function to obtain strenght
 *            from antecedents values. If output already assigned a rule strenght
 *            during inference use maximum function to determine
 *            final strength
 *
 ************************************************************/

rule_evaluation()
{
   struct rule_type* rule;
   struct rule_element_type* ip;    /* pointers */
   struct rule_element_type* tp;
   float strength;
   float nomatch = 0;
```

```
//Rule evaluation begins
        for (rule = Rule_Base; rule != NULL; rule = rule->next)
        {
          strength = UPPER_LIMIT;
          for (ip = rule->if_side; ip != NULL; ip = ip->next)
          {
            strength = min(strength, *(ip->value)); /*use min value */
          }
          for (tp = rule->then_side; tp != NULL; tp = tp->next)
          {
            *(tp->value) = max(strength, *(tp->value));      /* use max function */
            if (strength > 0)nomatch = 1;
          }
        }
        if (nomatch == 0) printf("NO MATCHING!\n");
      }     /* end of rule_evaluation() */

      /*************************************************************
       *
       *  FUNCTION NAME: defuzzification()
       *
       *  PARAMETERS: none
       *
       *  RETURN VALUE: crisp value that will modify skip threshold
       *
       *  DESCRIPTION:  Use Center of Gravity (COG) to calculates centroids for each
       *                membership function, limit each function height using
       *                applied rule strenght, compute areas or membership function,
       *                get sum of areas and sum of products to compute
       *               defuzzified  output value returned by fuzzy agent
       *
       *************************************************************/

      float defuzzification()
      {
        struct io_type* so;
        struct mf_type* mf;
        float sum_of_products;
        float sum_of_areas;
        float area, centroid;
        float outFIS;              /* defuzzified value */
        for (so = System_Output; so != NULL; so = so->next)
        {
          sum_of_products = 0;
          sum_of_areas = 0;
          for (mf = so->membership_functions; mf != NULL; mf = mf->next)
```

```
{
            area = compute_area_of_trapezoid(mf);
            centroid = mf->point1 + (mf->point2 - mf->point1) / 2;
            sum_of_products += area * centroid;
            sum_of_areas += area;
         }
         if (sum_of_areas == 0) {
            so->value = 0;
            return (so->value);
         }
         so->value = sum_of_products / sum_of_areas;
         outFIS = (so->value) / 1000;              /*return*/
         return(outFIS);
      }
   }       /* end of defuzzification () */


   /*****************************************************************
    *
    *  FUNCTION NAME:  compute_degree_of_membership()
    *
    *  PARAMETERS:     mf, input
    *
    *  RETURN VALUE:  void
    *
    *  DESCRIPTION:  Determine point1 ,
    *              determine point2 , and the
    *              slopes (calculated and stored during init_fuzzy).
    *
    *              delta1 = x - point1         delta2= point2-x
    *              MFdegree = min {(delta1 * Slope1) , (delta2*Slope2)}
    *
    ****************************************************************/

   compute_degree_of_membership(mf, input)
   struct mf_type* mf;
   float input;       // Compute degree begins
   {  float delta_1, delta_2;
      delta_1 = input - mf->point1;       // delta1 = x - point1
      delta_2 = mf->point2 - input;       // delta2 = point2 - x
       if ((delta_1 <= 0) || (delta_2 <= 0))mf->value = 0;
       else
       {
          mf->value = min((mf->slope1 * delta_1), (mf->slope2 * delta_2));
          mf->value = min(mf->value, UPPER_LIMIT);
       }
```

```
   for (j = 0; j < OUT_UNTS; j++) {   /*compare target with output*/
      patt_err[i][j] = (goals[i][j] - out_layer[j].output)
         * (goals[i][j] - out_layer[j].output);
      tot_patt_err[i] += patt_err[i][j];            /*sum errors*/

      /* calc delta_weight (n+1) */
      out_layer[j].delta = (goals[i][j] - out_layer[j].output) *
         (1.0 - out_layer[j].output) *
         out_layer[j].output;

      for (k = 0; k < HDN_UNTS + 1; k++) {/* backprop. learning rule for output*/
         out_layer[j].delta_weight[k] = learning_rate
            * out_layer[j].delta * hid_layer[k].output
            + (momentum * out_layer[j].delta_weight[k]);
      } /* Calculate all delta_weights feeding this output unit */
   } /* for all output units */

   for (j = 0; j < HDN_UNTS; j++) { /*hidden neurons = no target*/
      sum = 0.0;
      for (k = 0; k < OUT_UNTS; k++) {
         sum += out_layer[k].delta * out_layer[k].weight[j];
      }
      hid_layer[j].delta = sum * hid_layer[j].output
         * (1.0 - hid_layer[j].output);
      for (k = 0; k < IN_UNTS + 1; k++) { /* backprop. learning rule for hidden*/
         hid_layer[j].delta_weight[k] = learning_rate
            * hid_layer[j].delta * hid_layer[j].input[k]
            + (momentum * hid_layer[j].delta_weight[k]);
      }/* Calculate delta_weights on all inputs */
   }/* for all hidden units */

   /* ---------- NEW UPDATED WEIGHTS ---------- */
   for (j = 0; j < OUT_UNTS; j++) {
      for (k = 0; k < HDN_UNTS + 1; k++) {
         out_layer[j].weight[k] = (out_layer[j].weight[k] +
            out_layer[j].delta_weight[k]);
      } /* adjust all weights feeding this output unit */
   } /* for all output units */

   for (j = 0; j < HDN_UNTS; j++) {
      for (k = 0; k < IN_UNTS + 1; k++) {
         hid_layer[j].weight[k] += hid_layer[j].delta_weight[k];
      }/* adjust weights on all inputs */
   }/* for all hidden units */
}/* end of back_prop function */
```

```
/*************************************************************
 *
 *  FUNCTION NAME: train()
 *
 *  PARAMETERS: none
 *
 *  RETURN VALUE: void
 *
 *  DESCRIPTION:  training procedure => network trains with patterns
 *                with known outputs.
 *                 ***For FIL_BP, backward phase may be
 *                    interrupted when net should skip backpropagation()
 *                    function call
 *                 ** Keep track of savings (number of BP calls avoided
 *
 *
 *************************************************************/
void train()
{
   int j, k, iter = 0;
   int fwd_cnt = 0;
   int learn = 3;
   int examplar_cnt = 0;
   float ex_percentage = 0;
   float epoch_sys_err = 0;    /* for epoch's error*/
   float trend = 0;
   float previous_err = 0;     /* previous epoch's error*/
   float FISin1, FISin2;       /* to fuzzy agent  */
   float DeltaCE;
   do {
      sys_err = 0.0;                  /*setup system error */
      examplar_cnt = 0;       /*reset counter at the start of each iteration*/
      ex_percentage = 0;      /*reset counter at the start of each iteration*/
      for (j = 0; j < num_goals; j++) {
         propagate(j);              /* FWD function call */
         fwd_cnt++;
         learn = fuzzytutor(j);     //Fuzzy agent BP decision
         if (learn == 1) {      /* BWD function call */
            back_prop(j);                       /*backpropagation phase begins */
            sys_err += tot_patt_err[j];  /*accumulated error */
         }
         else {
            examplar_cnt++;
            sys_err += my_patt_err[j];  /*accumulated error */
         }
      } /* for all target patterns */
```

```
iter++;                    /*increment epochs counter */
    ex_percentage = (examplar_cnt * 100) / MX_PATTS;
    sys_err = 0.5 * sys_err / num_goals;
    epoch_sys_err = sys_err;              /* current epoch error  */
    trend = epoch_sys_err - previous_err;
    previous_err = epoch_sys_err;         /* previous Epoch error */
    FISin1 = epoch_sys_err;    /*To fuzzy agent */
    if (trend < 0) {
       FISin2 = trend * (-1);
    }
    else {
       FISin2 = trend;
    }
    float input1 = FISin1 * 100;              //normalize input values for FLE rules
    float input2 = FISin2 * 100;
   /*end of epoch, fuzzy inference system for control mechanism */
    init_fuzzy_system();             /* initialize fuzzy system */
    get_system_inputs(input1, input2);
    fuzzification();                 /* begin fuzzification of inputs  */
    rule_evaluation();               /* eval strenghts */
    DeltaCE = defuzzification();          /*defuzzified */
    report_FIS_system_outputs();
    if (iter > 100) {
       if (pat_err_tresh == 0) {
          printf("Pattern error ==zero  Skip_Treshold %f, \n", pat_err_tresh);
       }
       else {
          pat_err_tresh = pat_err_tresh - DeltaCE;
          if (pat_err_tresh < 0) {
             pat_err_tresh = 0;
          }
        }
    }
    if (sys_err < min_err_td) {                /*determine smallest error*/
       min_err_td = sys_err;
       min_err_iter = iter;
    }
    /*terminate training session */
} while ((sys_err > max_tot_err) && (iter < max_iterations));
if (sys_err > max_tot_err) {
    fprintf(rpt_ptr, "\nMaximum # of iterations was");
    fprintf(rpt_ptr, " exceeded, system did not converge");
}
else {
    fprintf(rpt_ptr, " \n The system converged.");
}
```

```
fprintf(rpt_ptr, "\n iterations = %d Sys_error = %g", iter, sys_err);
        fprintf(rpt_ptr,  "\n\t%d\t%g\t%g\t%d\t\t%d\t\t%d\n",  iter,  sys_err,  min_err_td,
min_err_iter, fwd_cnt, yes_bp_cnt, my_skip_cnt);
        fprintf(rpt_ptr, " \n\tPropagate() calls, (FWD passes) was %d\n", fwd_cnt);

        /*----record the final weights to file-------*/
        /* Loop for the hidden layer neuron's weights */
        fprintf(wtfin_ptr, "\n\n\n----------------------------------------------");
        fprintf(wtfin_ptr, "\n The learning rate and momentum are: %f,%f \t", learning_rate,
          momentum);
        fprintf(wtfin_ptr, "\nweights are:\n");
        fprintf(wtfin_ptr, "hidden_layer \n");
        for (j = 0; j < HDN_UNTS; j++) {        /* write the hidden layer's weights  */
          fprintf(wtfin_ptr, " \n");
          for (k = 0; k < IN_UNTS + 1; k++) {
            fprintf(wtfin_ptr, " %f \t\t", hid_layer[j].weight[k]);
          }/* weights on all inputs */
        }/* for all hidden units */
         /* Loop for the output layer neuron's weights */
        fprintf(wtfin_ptr, "\noutput_layer \n");
        for (j = 0; j < OUT_UNTS; j++) {        /* write the output layer's weights  */
          fprintf(wtfin_ptr, " \n");
          for (k = 0; k < HDN_UNTS + 1; k++) {    /* to a file */
            fprintf(wtfin_ptr, " %f \t\t", out_layer[j].weight[k]);
          } /*adjust all weights feeding this output unit */
        }/* for all output units */
      }  /* end of train function */

      /*************************************************************
       *
       *  FUNCTION NAME: process()
       *
       *  PARAMETERS: none
       *
       *  RETURN VALUE: void
       *
       *  DESCRIPTION:  processes pattern through the net &
       *              evaluate net
       *************************************************************/
      void process()
      {
        int i, j, k;
        float out_err;
        char choice;
  num_misses = 0;
```

```
   /*---------------- PROCESS TEST DATA ----------------------*/
   for (j = 0; j < num_samples; j++) {
      miss_flag = 0;
      propagate(j);       /*feedforward phase */
      for (k = 0; k < OUT_UNTS; k++) {
         if ((out_err = known_output[j][k] - out_layer[k].output) > 0.2) {
            miss_flag++;
         }
      }/* endloop (# of output units) */

    /*------------ following line for stats --------- */
      if (miss_flag > 0) num_misses++;
   }/* endloop (# of input sets) */
}/* end of process function */

/*************************************************************
 *
 *  FUNCTION NAME: evaluate()
 *
 *  PARAMETERS: none
 *
 *  RETURN VALUE: void
 *
 *  DESCRIPTION:  checks NN classification against known
 *              correct classification, for stats
 *************************************************************/
void evaluate()
{
   int correct = 0;
   int j, k;
   /*---------------- check  classifications produced -------*/
   fprintf(rpt_ptr, "\n\n THE PERFORMANCE RESULTS:\n");
   fprintf(rpt_ptr, "\n Number of testing samples=\t\t%d",
      num_samples);
   fprintf(rpt_ptr, "\n Number of correct classifications=\t%d",
      (correct = num_samples - num_misses));
   fprintf(rpt_ptr, "\n Percentage of correct classifications=\t %f",
      (100.0 * (float)correct) / ((float)num_samples));
   printf("\n....should now terminate MAIN\n");
}/* end of evaluate function

/*************************************************************
 *
 *  FUNCTION NAME: main
 *
*  PARAMETERS: none
```

```
 *
 *  RETURN VALUE: void
 *
 *  DESCRIPTION:  BPNN Learning algorithm with or without training
 *                optimization by assistance of fuzzy agent
 *
 ***********************************************************/
void main()
{
   char choice;
   void evaluate();
   void get_goals();
   void init_weights();
   void init_inputs();
   void process();
   void set_params();
   void train();
   float FISin1, FISin2;

   /*------- Open Weight Files ----------*/

      if ((wtfin_ptr = fopen("final_weights.txt", "w+")) == NULL) {
         exit(12);
       }   /* endif */

      if ((wt_ptr = fopen("initial_weights.txt", "w+")) == NULL) {
         exit(13);
      }/* endif */

   /*---- Full report file of network perfomance-------*/

      if ((rpt_ptr = fopen("performance_report.txt", "w+")) == NULL) {
       exit(18);
      }/* endif */

   /* Backpropagation Learning algorithm for NN begins */
   for (learning_rate = 0.1; learning_rate < 0.2; learning_rate += 0.1) {  /*Set parameters*/
      for (momentum = 0.1; momentum < 0.2; momentum += 0.1) {
         randseed = 568731L;
         min_err_td = 500.0;
         yes_bp_cnt = 0;
         my_skip_cnt = 0;
         pat_err_tresh = 0.0;        //Set treshold for FIL-BP mode
         init_weights();
         set_params();
```

```
 get_goals();
          train();
          /*------------ SET WEIGHTS AND THRESHOLDS ---------*/
          init_inputs();
          process();
          evaluate();
       }/*end for all momentum changes */
    }/*end for all learning rate changes */

    fclose(wt_ptr);                  /*close pointers*/
       fclose(wtfin_ptr);
} /***** end main ()   ************/
```

APPENDIX B: XOR Dataset Training Performance

XOR DATABASE																	
TRADITIONAL BACKPROPAGATION ALGORITHM										FUZZY-INFERENCE LEARNING (FIL) for BACKPROPAGATION ALGORITHM							
LEARNING PARAMETERS		TRAINING (Dataset size= 4)			PERFORMANCE			EVALUATION (Dataset size= 4)		TRAINING (Dataset size= 4)			PERFORMANCE			EVALUATION (Dataset size= 4)	
LEARNING RATE α	MOMENTUM μ	EPOCHS completed	FORWARD PHASE (# of function calls)	BACKWARD PHASE (# of BP function calls)	SAVINGS in BP (# of BP function calls skipped)	SAVINGS PERCENTAGE (% of BP skipped)	SYSTEM ERROR (MSE)	HITS	Classification Accuracy (%)	EPOCHS completed	FORWARD PHASE (# of function calls)	BACKWARD PHASE (# of BP function calls)	SAVINGS in BP (# of BP function calls skipped)	SAVINGS PERCENTAGE (% of BP skipped)	SYSTEM ERROR (MSE)	HITS	Classification Accuracy (%)
0.1	0.1	5000	20,000	20,000	0	0	0.0692636	2	50	5000	20,000	20,000	0	0	0.0692636	2	50
0.1	0.2	5000	20,000	20,000	0	0	0.0284018	2	50	5000	20,000	20,000	0	0	0.0284018	2	50
0.1	0.3	4970	19,880	19,880	0	0	0.0099954	4	100	4970	19,880	19,880	0	0	0.0099954	4	100
0.1	0.4	4255	17,020	17,020	0	0	0.00999923	4	100	4255	17,020	17,020	0	0	0.00999923	4	100
0.1	0.5	3544	14,176	14,176	0	0	0.00998602	4	100	3544	14,176	14,176	0	0	0.00998602	4	100
0.1	0.6	2834	11,336	11,336	0	0	0.00999374	4	100	2834	11,336	11,336	0	0	0.00999374	4	100
0.1	0.7	2126	8,504	8,504	0	0	0.00999622	4	100	2126	8,504	8,504	0	0	0.00999622	4	100
0.1	0.8	1420	5,680	5,680	0	0	0.00998652	4	100	1420	5,680	5,680	0	0	0.00998652	4	100
0.2	0.1	3211	12,844	12,844	0	0	0.00999732	4	100	3211	12,844	12,844	0	0	0.00999732	4	100
0.2	0.2	2850	11,400	11,400	0	0	0.00998131	4	100	2850	11,400	11,400	0	0	0.00998131	4	100
0.2	0.3	2489	9,956	9,956	0	0	0.00998233	4	100	2489	9,956	9,956	0	0	0.00998233	4	100
0.2	0.4	2129	8,516	8,516	0	0	0.00998163	4	100	2129	8,516	8,516	0	0	0.00998163	4	100
0.2	0.5	1770	7,080	7,080	0	0	0.0099984	4	100	1770	7,080	7,080	0	0	0.0099984	4	100
0.2	0.6	1414	5,656	5,656	0	0	0.00997628	4	100	1414	5,656	5,656	0	0	0.00997628	4	100
0.2	0.7	1060	4,240	4,240	0	0	0.00995706	4	100	1060	4,240	4,240	0	0	0.00995706	4	100
0.2	0.8	708	2,832	2,832	0	0	0.0099881	4	100	708	2,832	2,832	0	0	0.0099881	4	100
0.3	0.1	2242	8,968	8,968	0	0	0.00998948	4	100	2242	8,968	8,968	0	0	0.00998948	4	100
0.3	0.2	2008	8,032	8,032	0	0	0.00999923	4	100	2008	8,032	8,032	0	0	0.00999923	4	100
0.3	0.3	1759	7,036	7,036	0	0	0.00997097	4	100	1759	7,036	7,036	0	0	0.00997097	4	100
0.3	0.4	1498	5,992	5,992	0	0	0.00995347	4	100	1498	5,992	5,992	0	0	0.00995347	4	100
0.3	0.5	1235	4,940	4,940	0	0	0.00995587	4	100	1235	4,940	4,940	0	0	0.00995587	4	100
0.3	0.6	977	3,908	3,908	0	0	0.00997773	4	100	977	3,908	3,908	0	0	0.00997773	4	100
0.3	0.7	727	2,908	2,908	0	0	0.00995155	4	100	727	2,908	2,908	0	0	0.00995155	4	100
0.3	0.8	484	1,936	1,936	0	0	0.00996075	4	100	484	1,936	1,936	0	0	0.00996075	4	100
0.4	0.1	1813	7,252	7,252	0	0	0.00996423	4	100	1813	7,252	7,252	0	0	0.00996423	4	100
0.4	0.2	1599	6,396	6,396	0	0	0.00996047	4	100	1599	6,396	6,396	0	0	0.00996047	4	100
0.4	0.3	1397	5,588	5,588	0	0	0.00998891	4	100	1397	5,588	5,588	0	0	0.00998891	4	100
0.4	0.4	1203	4,812	4,812	0	0	0.00996439	4	100	1203	4,812	4,812	0	0	0.00996439	4	100
0.4	0.5	1013	4,052	4,052	0	0	0.0099346	4	100	1013	4,052	4,052	0	0	0.0099346	4	100
0.4	0.6	825	3,300	3,300	0	0	0.00992211	4	100	825	3,300	3,300	0	0	0.00992211	4	100
0.4	0.7	638	2,552	2,552	0	0	0.0099628	4	100	638	2,552	2,552	0	0	0.0099628	4	100
0.4	**0.8**	**453**	**1,812**	**1,812**	**0**	**0**	**0.00982161**	**4**	**100**	**453**	**1,812**	**1,810**	**2**	**0.110375276**	**0.00982187**	**4**	**100**
0.5	0.1	1388	5,552	5,552	0	0	0.00997564	4	100	1388	5,552	5,552	0	0	0.00997564	4	100
0.5	0.2	1230	4,920	4,920	0	0	0.00994353	4	100	1230	4,920	4,920	0	0	0.00994353	4	100
0.5	0.3	1075	4,300	4,300	0	0	0.0099637	4	100	1075	4,300	4,300	0	0	0.0099637	4	100
0.5	0.4	923	3,692	3,692	0	0	0.00994567	4	100	923	3,692	3,692	0	0	0.00994567	4	100
0.5	0.5	772	3,088	3,088	0	0	0.00995376	4	100	772	3,088	3,088	0	0	0.00995376	4	100
0.5	0.6	622	2,488	2,488	0	0	0.0099073	4	100	622	2,488	2,488	0	0	0.0099073	4	100
0.5	0.7	471	1,884	1,884	0	0	0.00994222	4	100	471	1,884	1,884	0	0	0.00994222	4	100
0.5	0.8	320	1,280	1,280	0	0	0.00979552	4	100	320	1,280	1,280	0	0	0.00979552	4	100
0.6	0.1	1141	4,564	4,564	0	0	0.00997499	4	100	1141	4,564	4,564	0	0	0.00997499	4	100
0.6	0.2	1011	4,044	4,044	0	0	0.00999485	4	100	1011	4,044	4,044	0	0	0.00999485	4	100
0.6	0.3	884	3,536	3,536	0	0	0.0099513	4	100	884	3,536	3,536	0	0	0.0099513	4	100
0.6	0.4	758	3,032	3,032	0	0	0.00993738	4	100	758	3,032	3,032	0	0	0.00993738	4	100
0.6	0.5	633	2,532	2,532	0	0	0.00991306	4	100	633	2,532	2,532	0	0	0.00991306	4	100
0.6	0.6	508	2,032	2,032	0	0	0.00995125	4	100	508	2,032	2,032	0	0	0.00995125	4	100
0.6	0.7	384	1,536	1,536	0	0	0.0098677	4	100	384	1,536	1,536	0	0	0.0098677	4	100
0.6	0.8	260	1,040	1,040	0	0	0.00975755	4	100	260	1,040	1,040	0	0	0.00975755	4	100
0.7	0.1	974	3,896	3,896	0	0	0.00999943	4	100	974	3,896	3,896	0	0	0.00999943	4	100
0.7	0.2	864	3,456	3,456	0	0	0.00992723	4	100	864	3,456	3,456	0	0	0.00992723	4	100
0.7	0.3	754	3,016	3,016	0	0	0.00997879	4	100	754	3,016	3,016	0	0	0.00997879	4	100
0.7	0.4	646	2,584	2,584	0	0	0.00995529	4	100	646	2,584	2,584	0	0	0.00995529	4	100
0.7	0.5	539	2,156	2,156	0	0	0.00990377	4	100	539	2,156	2,156	0	0	0.00990377	4	100
0.7	0.6	432	1,728	1,728	0	0	0.00992859	4	100	432	1,728	1,728	0	0	0.00992859	4	100
0.7	0.7	326	1,304	1,304	0	0	0.00985901	4	100	326	1,304	1,304	0	0	0.00985901	4	100
0.7	0.8	220	880	880	0	0	0.00995727	4	100	220	880	880	0	0	0.00995727	4	100
0.8	0.1	854	3,416	3,416	0	0	0.00991832	4	100	854	3,416	3,416	0	0	0.00991832	4	100
0.8	0.2	756	3,024	3,024	0	0	0.00995913	4	100	756	3,024	3,024	0	0	0.00995913	4	100
0.8	0.3	660	2,640	2,640	0	0	0.00993914	4	100	660	2,640	2,640	0	0	0.00993914	4	100
0.8	0.4	565	2,260	2,260	0	0	0.00991827	4	100	565	2,260	2,260	0	0	0.00991827	4	100
0.8	0.5	471	1,884	1,884	0	0	0.00986389	4	100	471	1,884	1,884	0	0	0.00986389	4	100
0.8	0.6	377	1,508	1,508	0	0	0.00991234	4	100	377	1,508	1,508	0	0	0.00991234	4	100
0.8	0.7	284	1,136	1,136	0	0	0.00990903	4	100	284	1,136	1,136	0	0	0.00990903	4	100
0.8	0.8	192	768	768	0	0	0.00990392	4	100	192	768	768	0	0	0.00990392	4	100

APPENDIX C: Iris Dataset Training Performance

		IRIS DATABASE															
		TRADITIONAL BACKPROPAGATION ALGORITHM								FUZZY-INFERENCE LEARNING (FIL) for BACKPROPAGATION ALGORITHM							
LEARNING PARAMETERS		TRAINING (Dataset size= 90)			PERFORMANCE			EVALUATION (Dataset size= 60)		TRAINING (Dataset size= 90)			PERFORMANCE			EVALUATION (Dataset size= 60)	
LEARNING RATE α	MOMENTUM μ	EPOCHS completed	FORWARD PHASE (# of function calls)	BACKWARD PHASE (# of BP function calls)	SAVINGS in BP (# of BP function calls skipped)	SAVINGS PERCENTAGE (% of BP skipped)	SYSTEM ERROR (MSE)	HITS	Classification Accuracy (%)	EPOCHS completed	FORWARD PHASE (# of function calls)	BACKWARD PHASE (# of BP function calls)	SAVINGS in BP (# of BP function calls skipped)	SAVINGS PERCENTAGE (% of BP skipped)	SYSTEM ERROR (MSE)	HITS	Classification Accuracy (%)
0.1	0.1	5000	450,000	450,000	0	0	0.0108802	58	96.67	5000	450,000	79,511	370489	82.33	0.0132188	58	96.67
0.1	0.2	5000	450,000	450,000	0	0	0.0109005	58	96.67	5000	450,000	80,045	369955	82.21	0.0135242	58	96.67
0.1	0.3	5000	450,000	450,000	0	0	0.0109405	58	96.67	5000	450,000	82,739	367261	81.61	0.0136776	58	96.67
0.1	0.4	5000	450,000	450,000	0	0	0.010956	58	96.67	5000	450,000	86,085	363915	80.87	0.0136543	58	96.67
0.1	0.5	5000	450,000	450,000	0	0	0.0108934	58	96.67	5000	450,000	92,835	357165	79.37	0.0136508	58	96.67
0.1	0.6	5000	450,000	450,000	0	0	0.0109196	58	96.67	5000	450,000	92,883	357117	79.36	0.0135982	58	96.67
0.1	0.7	5000	450,000	450,000	0	0	0.0101352	58	96.67	5000	450,000	77,240	372760	82.84	0.0133072	58	96.67
0.1	0.8	5000	450,000	450,000	0	0	0.0108729	56	93.33	5000	450,000	220,996	229004	50.89	0.010012	58	96.67
0.2	0.1	5000	450,000	450,000	0	0	0.0109054	58	96.67	5000	450,000	87,971	362029	80.45	0.0136392	58	96.67
0.2	0.2	5000	450,000	450,000	0	0	0.0108929	58	96.67	5000	450,000	85,809	364191	80.93	0.0135776	58	96.67
0.2	0.3	5000	450,000	450,000	0	0	0.0108836	58	96.67	5000	450,000	85,463	364537	81.01	0.0135585	58	96.67
0.2	0.4	5000	450,000	450,000	0	0	0.0108744	58	96.67	5000	450,000	86,023	363977	80.88	0.0134786	58	96.67
0.2	0.5	5000	450,000	450,000	0	0	0.0108417	58	96.67	5000	450,000	84,188	365812	81.29	0.0134424	58	96.67
0.2	0.6	5000	450,000	450,000	0	0	0.0108254	58	96.67	5000	450,000	85,386	364614	81.03	0.0135685	58	96.67
0.2	0.7	5000	450,000	450,000	0	0	0.0100547	58	96.67	5000	450,000	271,352	178648	39.70	0.0108519	58	96.67
0.2	0.8	5000	450,000	450,000	0	0	0.0108036	58	96.67	5000	450,000	288,343	161657	35.92	0.173318	17	28.33
0.3	0.1	5000	450,000	450,000	0	0	0.0108629	58	96.67	5000	450,000	87,537	362463	80.55	0.0134366	58	96.67
0.3	0.2	5000	450,000	450,000	0	0	0.010861	58	96.67	5000	450,000	84,225	365775	81.28	0.0133323	58	96.67
0.3	0.3	5000	450,000	450,000	0	0	0.0108403	58	96.67	5000	450,000	85,580	364420	80.98	0.0130637	58	96.67
0.3	0.4	5000	450,000	450,000	0	0	0.0103046	58	96.67	5000	450,000	84,582	365418	81.20	0.0137896	58	96.67
0.3	0.5	5000	450,000	450,000	0	0	0.0108479	58	96.67	5000	450,000	79,504	370496	82.33	0.0137679	58	96.67
0.3	0.6	5000	450,000	450,000	0	0	0.0108083	58	96.67	5000	450,000	276,221	173779	38.62	0.0108554	58	96.67
0.3	0.7	5000	450,000	450,000	0	0	0.0108049	58	96.67	5000	450,000	438,162	11838	2.63	0.0108098	58	96.67
0.3	0.8	5000	450,000	450,000	0	0	0.169952	17	28.33	5000	450,000	286,261	163739	36.39	0.180452	17	28.33
0.4	0.1	5000	450,000	450,000	0	0	0.0108406	58	96.67	5000	450,000	88,729	361271	80.28	0.0132535	55	91.67
0.4	0.2	5000	450,000	450,000	0	0	0.0103098	58	96.67	5000	450,000	87,528	362472	80.55	0.0139092	58	96.67
0.4	0.3	5000	450,000	450,000	0	0	0.0100628	58	96.67	5000	450,000	83,809	366191	81.38	0.0139372	58	96.67
0.4	0.4	5000	450,000	450,000	0	0	0.0101383	58	96.67	5000	450,000	237,059	212941	47.32	0.0108288	58	96.67
0.4	0.5	5000	450,000	450,000	0	0	0.0108115	58	96.67	5000	450,000	254,301	195699	43.49	0.0108179	58	96.67
0.4	0.6	3691	332,190	332,190	0	0	0.0099999	58	96.67	5000	450,000	317,777	132223	29.38	0.0108147	58	96.67
0.4	0.7	5000	450,000	450,000	0	0	0.177426	17	28.33	5000	450,000	286,707	163293	36.29	0.184786	17	28.33
0.4	0.8	5000	450,000	450,000	0	0	0.188408	17	28.33	5000	450,000	285,894	164106	36.47	0.186833	17	28.33
0.5	0.1	4820	433,800	433,800	0	0	0.00999996	58	96.67	5000	450,000	83,730	366270	81.39	0.0139631	58	96.67
0.5	0.2	5000	450000	450000	0	0	0.0102075	58	96.67	5000	450,000	294,422	155578	34.57	0.0103048	58	96.67
0.5	0.3	5000	450000	450000	0	0	0.0105632	58	96.67	5000	450,000	85,292	364708	81.05	0.0132817	58	96.67
0.5	0.4	5000	450000	450000	0	0	0.0101864	58	96.67	5000	450,000	282,761	167239	37.16	0.0108034	58	96.67
0.5	0.5	5000	450000	450000	0	0	0.0103876	58	96.67	5000	450,000	438,606	11394	2.53	0.0108126	58	96.67
0.5	0.6	5000	450000	450000	0	0	0.010803	59	98.33	5000	450,000	427,885	22115	4.91	0.0107966	59	98.33
0.5	0.7	5000	450000	450000	0	0	0.181578	17	28.33	5000	450,000	286,593	163407	36.31	0.181376	17	28.33
0.5	0.8	5000	450000	450000	0	0	0.177435	58	96.67	5000	450,000	285,264	164736	36.61	0.193164	17	28.33
0.6	0.1	4198	377820	377820	0	0	0.00999996	58	96.67	5000	450,000	81,216	368784	81.95	0.0138536	58	96.67
0.6	0.2	5000	450000	450000	0	0	0.010147	58	96.67	5000	450,000	84,367	365633	81.25	0.0132746	58	96.67
0.6	0.3	5000	450000	450000	0	0	0.0101985	58	96.67	4812	433,080	315,228	117852	27.21	0.0099999	58	96.67
0.6	0.4	5000	450000	450000	0	0	0.010906	58	96.67	5000	450,000	297,066	152934	33.99	0.0108076	58	96.67
0.6	0.5	5000	450000	450000	0	0	0.0107977	58	96.67	5000	450,000	443,884	6116	1.36	0.0108006	58	96.67
0.6	0.6	5000	450000	450000	0	0	0.195139	17	28.33	5000	450,000	286,446	163554	36.35	0.178111	17	28.33
0.6	0.7	5000	450000	450000	0	0	0.173092	17	28.33	5000	450,000	286,102	163898	36.42	0.196224	17	28.33
0.6	0.8	5000	450000	450000	0	0	0.355858	0	0.00	5000	450,000	286,611	163389	36.31	0.180137	17	28.33
0.7	0.1	5000	450000	450000	0	0	0.0108181	58	96.67	5000	450,000	274,858	175142	38.92	0.0102587	58	96.67
0.7	0.2	4842	435780	435780	0	0	0.00999996	58	96.67	4188	376,920	215,429	161491	42.84	0.01	58	96.67
0.7	0.3	5000	450000	450000	0	0	0.0101927	58	96.67	4533	407,970	198,381	209589	51.37	0.01	58	96.67
0.7	0.4	5000	450000	450000	0	0	0.0100972	58	96.67	5000	450,000	438,733	11267	2.50	0.0107999	58	96.67
0.7	0.5	5000	450000	450000	0	0	0.0107982	58	96.67	5000	450,000	426,596	23404	5.20	0.0108632	58	96.67
0.7	0.6	5000	450000	450000	0	0	0.171049	17	28.33	5000	450,000	285,749	164251	36.50	0.189843	17	28.33
0.7	0.7	5000	450000	450000	0	0	0.175285	17	28.33	5000	450,000	285,560	164440	36.54	0.176342	17	28.33
0.7	0.8	5000	450000	450000	0	0	0.209634	17	28.33	5000	450,000	305,129	144871	32.19	0.20642	17	28.33
0.8	0.1	5000	450000	450000	0	0	0.0108037	58	96.67	5000	450,000	261,792	188208	41.82	0.0101819	58	96.67
0.8	0.2	5000	450000	450000	0	0	0.0108045	58	96.67	5000	450,000	231,172	218828	48.63	0.0100521	58	96.67
0.8	0.3	5000	450000	450000	0	0	0.0103872	58	96.67	5000	450,000	439,884	10116	2.25	0.0107964	58	96.67
0.8	0.4	5000	450000	450000	0	0	0.0107963	59	98.33	5000	450,000	441,713	8287	1.84	0.0102112	58	96.67
0.8	0.5	5000	450000	450000	0	0	0.188219	17	28.33	5000	450,000	324,679	125321	27.85	0.169913	17	28.33
0.8	0.6	5000	450000	450000	0	0	0.18431	17	28.33	5000	450,000	285,713	164287	36.51	0.193656	17	28.33
0.8	0.7	5000	450000	450000	0	0	0.177427	17	28.33	5000	450,000	285,394	164606	36.58	0.177557	17	28.33
0.8	0.8	5000	450000	450000	0	0	0.217935	41	68.33	5000	450,000	424,318	25682	5.71	0.386947	0	0.00

APPENDIX D: Connectionist Bench (Vowel Recognition) Training Performance

VOWEL DATABASE																	
TRADITIONAL BACKPROPAGATION ALGORITHM										FUZZY-INFERENCE LEARNING (FIL) for BACKPROPAGATION ALGORITHM							
LEARNING PARAMETERS		TRAINING (Dataset size= 528)			PERFORMANCE			EVALUATION (Dataset size= 462)		TRAINING (Dataset size= 528)			PERFORMANCE			EVALUATION (Dataset size= 462)	
LEARNING RATE α	MOMENTUM μ	EPOCHS completed	FORWARD PHASE (# of function calls)	BACKWARD PHASE (# of BP function calls)	SAVINGS in BP (# of BP function calls skipped)	SAVINGS PERCENTAGE (% of BP skipped)	SYSTEM ERROR (MSE)	HITS	Classification Accuracy (%)	EPOCHS completed	FORWARD PHASE (# of function calls)	BACKWARD PHASE (# of BP function calls)	SAVINGS in BP (# of BP function calls skipped)	SAVINGS PERCENTAGE (% of BP skipped)	SYSTEM ERROR (MSE)	HITS	Classification Accuracy (%)
0.1	0.1	5000	2640000	2640000	0	0	0.05454	143	30.95	5000	2640000	1322489	1317511	49.9057197	0.061372	143	30.95
0.1	0.2	5000	2640000	2640000	0	0	0.040294	133	28.79	5000	2640000	1251542	1388458	52.59310606	0.042329	122	26.41
0.1	0.3	5000	2640000	2640000	0	0	0.053621	164	35.50	5000	2640000	1168205	1471795	55.74981061	0.055448	164	35.50
0.1	0.4	5000	2640000	2640000	0	0	0.053102	165	35.71	5000	2640000	1073158	1566842	59.35007576	0.054124	167	36.15
0.1	0.5	5000	2640000	2640000	0	0	0.056532	178	38.53	5000	2640000	1027958	1612042	61.06219697	0.055935	167	36.15
0.1	0.6	5000	2640000	2640000	0	0	0.067305	161	34.85	5000	2640000	991210	1648790	62.45416667	0.049761	135	29.22
0.1	0.7	5000	2640000	2640000	0	0	0.048648	156	33.77	5000	2640000	1032078	1607922	60.90613636	0.059415	162	35.06
0.1	0.8	5000	2640000	2640000	0	0	0.051685	169	36.58	5000	2640000	796815	1843185	69.81761364	0.046624	164	35.50
0.2	0.1	5000	2640000	2640000	0	0	0.037622	180	38.96	5000	2640000	1057995	1582005	59.92443182	0.042602	173	37.45
0.2	0.2	5000	2640000	2640000	0	0	0.037809	174	37.66	5000	2640000	970410	1669590	63.24204545	0.038556	184	39.83
0.2	0.3	5000	2640000	2640000	0	0	0.037457	183	39.61	5000	2640000	928855	1711145	64.81609848	0.056184	182	39.39
0.2	0.4	5000	2640000	2640000	0	0	0.043656	149	32.25	5000	2640000	885472	1754528	66.45939394	0.047465	168	36.36
0.2	0.5	5000	2640000	2640000	0	0	0.081694	154	33.33	5000	2640000	844497	1795503	68.01147727	0.050865	157	33.98
0.2	0.6	5000	2640000	2640000	0	0	0.059265	163	35.28	5000	2640000	905905	1734095	65.68541667	0.060129	154	33.33
0.2	0.7	5000	2640000	2640000	0	0	0.048356	185	40.04	5000	2640000	736069	1903931	72.11859848	0.06048	165	35.71
0.2	0.8	5000	2640000	2640000	0	0	0.04907	193	41.77	5000	2640000	657081	1982919	75.11056818	0.058298	159	34.42
0.3	0.1	5000	2640000	2640000	0	0	0.042746	186	40.26	5000	2640000	898561	1741439	65.96359848	0.043935	190	41.13
0.3	0.2	5000	2640000	2640000	0	0	0.044322	161	34.85	5000	2640000	869065	1770935	67.08087121	0.042676	182	39.39
0.3	0.3	5000	2640000	2640000	0	0	0.047982	181	39.18	5000	2640000	854997	1785003	67.61375	0.044939	172	37.23
0.3	0.4	5000	2640000	2640000	0	0	0.080224	163	35.28	5000	2640000	1077924	1562076	59.16954545	0.075077	167	36.15
0.3	0.5	5000	2640000	2640000	0	0	0.044818	181	39.18	5000	2640000	777018	1862982	70.5675	0.039517	199	43.07
0.3	0.6	5000	2640000	2640000	0	0	0.069346	158	34.20	5000	2640000	910710	1729290	65.50340909	0.075909	149	32.25
0.3	0.7	5000	2640000	2640000	0	0	0.048169	209	45.24	5000	2640000	641525	1998475	75.69981061	0.100621	182	39.39
0.3	0.8	5000	2640000	2640000	0	0	0.059782	167	36.15	5000	2640000	2462621	177379	6.718901515	0.0565	208	45.02
0.4	0.1	5000	2640000	2640000	0	0	0.043404	186	40.26	5000	2640000	720451	1919549	72.71018939	0.034564	201	43.51
0.4	0.2	5000	2640000	2640000	0	0	0.05328	183	39.61	5000	2640000	871893	1768107	66.97375	0.048902	201	43.51
0.4	0.3	5000	2640000	2640000	0	0	0.039098	170	36.80	5000	2640000	733259	1906741	72.22503788	0.048319	141	30.52
0.4	0.4	5000	2640000	2640000	0	0	0.042623	168	36.36	5000	2640000	801867	1838133	69.62625	0.037828	213	46.10
0.4	0.5	5000	2640000	2640000	0	0	0.051302	193	41.77	5000	2640000	711450	1928550	73.05113636	0.042452	199	43.07
0.4	0.6	5000	2640000	2640000	0	0	0.081259	180	38.96	5000	2640000	658494	1981506	75.05704545	0.060517	174	37.66
0.4	0.7	5000	2640000	2640000	0	0	0.0454	177	38.31	5000	2640000	679553	1960447	74.25935606	0.057966	174	37.66
0.4	0.8	5000	2640000	2640000	0	0	0.139407	160	34.63	5000	2640000	1451268	1188732	45.02772727	0.12537	185	40.04
0.5	0.1	5000	2640000	2640000	0	0	0.042982	193	41.77	5000	2640000	745315	1894685	71.76837121	0.050192	159	34.42
0.5	0.2	5000	2640000	2640000	0	0	0.059	170	36.80	5000	2640000	804670	1835330	69.52007576	0.032297	217	46.97
0.5	0.3	5000	2640000	2640000	0	0	0.047817	207	44.81	5000	2640000	822146	1817854	68.85810606	0.035916	226	48.92
0.5	0.4	5000	2640000	2640000	0	0	0.027072	188	40.69	5000	2640000	646381	1993619	75.51587121	0.03398	186	40.26
0.5	0.5	5000	2640000	2640000	0	0	0.046632	201	43.51	5000	2640000	695035	1944965	73.67291667	0.065341	179	38.74
0.5	0.6	5000	2640000	2640000	0	0	0.075476	182	39.39	5000	2640000	792662	1847338	69.97492424	0.059684	206	44.59
0.5	0.7	5000	2640000	2640000	0	0	0.082531	207	44.81	5000	2640000	706765	1933235	73.22859848	0.082346	169	36.58
0.5	0.8	5000	2640000	2640000	0	0	0.191958	170	36.80	5000	2640000	2536024	103976	3.938484848	0.104451	217	46.97
0.6	0.1	5000	2640000	2640000	0	0	0.049775	191	41.34	5000	2640000	1015848	1624152	61.52090909	0.079954	173	37.45
0.6	0.2	5000	2640000	2640000	0	0	0.039743	209	45.24	5000	2640000	682628	1957372	74.14287879	0.037048	204	44.16
0.6	0.3	5000	2640000	2640000	0	0	0.044661	173	37.45	5000	2640000	718693	1921307	72.7767803	0.042471	173	37.45
0.6	0.4	5000	2640000	2640000	0	0	0.058142	159	34.42	5000	2640000	741432	1898568	71.91545455	0.040731	177	38.31
0.6	0.5	5000	2640000	2640000	0	0	0.091047	169	36.58	5000	2640000	988965	1651035	62.53920455	0.065607	197	42.64
0.6	0.6	5000	2640000	2640000	0	0	0.12559	157	33.98	5000	2640000	869656	1770344	67.05848485	0.057751	175	37.88
0.6	0.7	5000	2640000	2640000	0	0	0.061358	158	34.20	5000	2640000	1058993	1581007	59.88662879	0.108105	173	37.45
0.6	0.8	5000	2640000	2640000	0	0	0.208593	144	31.17	5000	2640000	2252213	387787	14.68890152	0.144434	183	39.61
0.7	0.1	5000	2640000	2640000	0	0	0.050605	168	36.36	5000	2640000	838372	1801628	68.24348485	0.063205	167	36.15
0.7	0.2	5000	2640000	2640000	0	0	0.05306	188	40.69	5000	2640000	864878	1775122	67.2394697	0.048077	197	42.64
0.7	0.3	5000	2640000	2640000	0	0	0.051471	177	38.31	5000	2640000	2383447	256553	9.717916667	0.073561	161	34.85
0.7	0.4	5000	2640000	2640000	0	0	0.078078	165	35.71	5000	2640000	947766	1692234	64.09977273	0.066369	177	38.31
0.7	0.5	5000	2640000	2640000	0	0	0.094624	157	33.98	5000	2640000	2366454	273546	10.36159091	0.090811	169	36.58
0.7	0.6	5000	2640000	2640000	0	0	0.105275	188	40.69	5000	2640000	901841	1738159	65.83935606	0.069057	187	40.48
0.7	0.7	5000	2640000	2640000	0	0	0.173572	127	27.49	5000	2640000	2418526	221474	8.389166667	0.103059	173	37.45
0.7	0.8	5000	2640000	2640000	0	0	0.262952	103	22.29	5000	2640000	1701343	938657	35.55518939	0.240363	112	24.24
0.8	0.1	5000	2640000	2640000	0	0	0.092734	189	40.91	5000	2640000	911863	1728137	65.45973485	0.069028	176	38.10
0.8	0.2	5000	2640000	2640000	0	0	0.062381	171	37.01	5000	2640000	722433	1917567	72.63511364	0.0894	164	35.50
0.8	0.3	5000	2640000	2640000	0	0	0.059794	209	45.24	5000	2640000	862115	1777885	67.34412879	0.07728	180	38.96
0.8	0.4	5000	2640000	2640000	0	0	0.087366	192	41.56	5000	2640000	2303733	336267	12.73738636	0.075231	205	44.37
0.8	0.5	5000	2640000	2640000	0	0	0.106001	162	35.06	5000	2640000	1251741	1388259	52.58556818	0.107868	143	30.95
0.8	0.6	5000	2640000	2640000	0	0	0.120052	179	38.74	5000	2640000	2264673	375327	14.21693182	0.070865	184	39.83
0.8	0.7	5000	2640000	2640000	0	0	0.163065	157	33.98	5000	2640000	2119935	520065	19.69943182	0.207598	182	39.39
0.8	0.8	5000	2640000	2640000	0	0	0.240235	141	30.52	5000	2640000	2522182	117818	4.46280303	0.096566	174	37.66

VITA

Miroslava Barúa received her Bachelor of Science degree in Electrical Engineering from The University of Texas at El Paso in December 2000. In 2004, she received her Master of Science degree in Electrical Engineering from The University of Texas at El Paso earning the recognition for Outstanding Thesis in Computer Engineering Honors Award. She later joined the doctoral program in Electrical and Computer Engineering at The University of Texas at El Paso and worked under the guidance and mentorship of Dr. Patricia Nava. Her teaching experience includes instruction and support for courses in the area of Digital System Design I and II, Foundations of Deep Learning, Microprocessor Systems, Senior Project Laboratory and Seminar of Critical Inquiry focused in Foundations of Engineering. She has received the Outstanding Teaching Award at the Doctoral level from UTEP Graduate School and several teaching certifications from the Center for Instructional Design at UTEP. Her research and professional experience include working on projects with Neuro-Fuzzy Systems Research Group, Research Institute for Manufacturing and Engineering Systems, Academic Affairs and Undergraduate Studies for College of Engineering, Software Development Engineering of Intelligent Agents for Supply Chain Management and Assurance, White Sands Missile Range Data Support Division, and Institute of Defense and Security. Her research interests are in Artificial Intelligence, Deep Learning, Neural Network Architectures, and Digital Systems. She is a member of Tau Beta Pi Engineering Honor Society, IEEE- Eta Kappa Nu Honor Society, Women In Engineering and Society of Women Engineers. She has presented her research work at national and international conference meetings and has several publications. Some of her publications are the following:

1. **M. Barua** and P. Nava, "Trends and strategies to optimize training processes to decrease learning time in Deep Neural Networks," 2022 World Congress in Computer Science, Computer Engineering and Applied Computing (CSCE'22), July 25-28, 2022. In press for publication in Springer Book Series: Transactions on Computational Science & Computational Intelligence (Hamid R. Arabnia, Ed.) Electronic ISSN: 2569-7080, Print ISSN: 2569-7072.

2. R. Villegas, P. Nava, and **M. Barua**, "Data Mining-based Techniques in Critical Operation of Electrical Transmission and Distribution Systems in a Natural Disaster Event: Future Direction Review," *Proceedings of the 13th Annual IEEE International Systems Conference* (SYSCON 2019), pp. 888-895, ISBN 978-1-5386-8396-5, March 2019.
3. Virani, S., Burnham, I. B., Gonzalez, V., **Barua, M.**, Andrade, S. J., "Work in Progress: Designing an Innovative Curriculum for Engineering in High School (ICE-HS)", *American Society for Engineering Education (ASEE) Annual Conference & Exposition*, Vancouver, BC. DOI 10.18260/1-2—18773, ISSN 2153-5965, pp. 22.1701.1 - 22.1701.11, June, 2011
4. N. Kilicay-Ergin, **M. Barua**, R. Pineda, "Prognostics Health Management Process Framework for System-of-Systems", *Intelligent Engineering Systems through Artificial Neural Networks (ANNIE 2010),* Volume 20. Ed. Dagli, CH. ASME Press, 2010.
5. **M. Barua**, "Hardware Implementation of Radial Basis Function Networks using Field Programmable Gate Arrays", *Proceedings of the Live-Virtual-Constructive Conference (LVCC), International Test and Evaluation Association (ITEA 2010),* El Paso, Texas, January 11-14, 2010
6. H. Nazeran, M, Goldman, P. Nava, B. Diong, **M. Barua**, and A. Crockett, "Forced Oscillation: Neural Networks Can Advance the Utility of Impulse Oscillometry in Assessment of Lung Function in Children", *International Journal of Medical Implants and Devices*, Vol. 3, No. 3 pp. 139-159, 2007.
7. **M. Barua**, H. Nazeran, P. Nava, B. Diong, and M. Goldman, "Implementation of Artificial Neural Networks to Classify Impulse Oscillometric Patterns of Lung Function in Asthmatic Children," *Proceedings of the 5th International Workshop on Biomedical Signal Interpretation*, Tokyo, Japan, September 6-8, 2005.
8. **M. Barua**, H. Nazeran, P. Nava, B. Diong, and M. Goldman, "Classification of Impulse Oscillometric Patterns of Lung Function in Asthmatic Children Using Artificial Neural Networks," *Proceedings of the 2005 27th Annual International Conference of the Engineering in Medicine and Biology Society, IEEE-EMBS 2005,* pp. 327-331, 2005.
9. **M. Barua**, H. Nazeran, P. Nava, and V. Granda, "Classification of Pulmonary Diseases using Artificial Neural Networks," *Intelligent Engineering Systems through Artificial Neural Networks (ANNIE 2004)*, Vol. 14, pp. 755-760, 2004.
10. **M. Barua**, H. Nazeran, P. Nava, V. Granda, and B. Diong, "Classification of Pulmonary Diseases by an Artificial Neural Network, Based on Measurements from the Impulse Oscillometry System," *Proc. Of the 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 3848-3851, 2004.

Contact Information: miroslav@miners.utep.edu