Addis Ababa University
College of Natural Sciences

Lightweight Neural Networks for Context Aware Autonomous Embedded System Development

Abdi Mosisa Dera

A Thesis Submitted to the Department of Computer Science in Partial Fulfilment for the Degree of Master of Science in Computer Science

Addis Ababa, Ethiopia

February 2020

Addis Ababa University

College of Natural Sciences

Abdi MosisaDera

Advisor: Dr. Dagmawi Lemma Gobena

This is to certify that the thesis prepared by Abdi Mosisa Dera, titled: *Lightweight Neural Networks for Context Aware Autonomous Embedded System Development* and submitted in partial fulfilment of the requirements for the Degree of Master of Science in Computer Science complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the Examining Committee:

Name Signature Date

Advisor:_____

Examiner:_____

Examiner:_____

Abstract

An embedded system is a microcontroller or microprocessor-based system which is designed to perform a specific task by collecting, processing and communicating information. While focusing on specific task, it is also desired to make such system for better and efficient result. In due course, one of the challenges is contextualizing the collected information to predict the output and making smart decision to produce the output. The learning system that can contextualize the surrounding environment should have a capability of automatic mechanism of inferring information like humans do. This calls for neural networks that provide an embedded intelligence for smart systems to make decisions at machine speed. The main challenge to develop such system is the constraints in memory size, computational power and other characteristics of embedded system that can significantly restrict developers from implementing learning algorithms to solve the problem. This thesis presents lightweight neural networks so as to show a method for implementing context-aware embedded system in environment where there is resource limitation. A testbed is setup for collecting the data, training and evaluation. Arduino board is investigated as a main experimental device for the proposed algorithms. The algorithms are simulated using C on Arduino. A good result was obtained after deploying the algorithm and knowledgebase on arduino board for sensor reading.

Key words: Embedded Systems, Context awareness, Neural Networks

Dedication

This work is dedicated to my families for their support, encouragement, and patience in my work and throughout my life.

Acknowledgements

Success is never achieved single handed. So, it is my duty to acknowledge all those who have provided helping hands in making this thesis success. Firstly I would like to thank ALMIGHTY GOD for supporting me in every way of my life. I would also like to express my deep sense of gratitude towards my research advisor Dr. **Dagmawi Lemma** for his invaluable guidance, co-operation and approval by him in successful completion of this work. Secondly, I am privileged to express my sense of gratitude to my respected instructors whose unparalleled knowledge, moral fiber and judgment along with their know-how, was an immense support in completing this thesis. Those who in one way or another have contributed to the success of the thesis are also recognized. Last but not least, a great deal of appreciation and best wishes to all my friends for their encouragement during this work.

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
BN	Bayesian Network
CPT	Conditional Probability Table
DAG	Direct Acyclic Graph
DT	Decision Tree
ES	Embedded System
HMM	Hidden Markov Model
IoT	Internet of Things
ML	Machine Learning
MLP	Machine Learning Perception
MSE	Mean Square Error
OWL	Web Ontology Language

1. Introduction

1.1 Background

An embedded system is a system which is often implemented in microcontroller to perform a specific task. It usually contains input receptors, a processor that process the input data or what has been stored in the memory and/or process the input, and an actuator that makes an output based on the processed inputs as per the embedded code in the device. The use of embedded system is multi-dimensional ranging from industrial machines to home appliances. It can also play significant role in building smart computing environment that mainly rely on context awareness [4].

Context awareness enables a system to understand and react to certain conditions based on certain contextual information. Such system often uses embedded devices to capture the surrounding information. Context-aware embedded system is beyond self-aware computing in which self-aware computing is about making a system knowledgeable about its internal operations, states and process whereas, context aware system is incorporating global-level awareness about the system and its environment [5]. Various algorithms need to be applied in embedded systems to make it context aware system. The existing embedded system algorithms consist of mostly rule based Artificial Intelligence (AI) algorithms such as expert systems. These algorithms lack flexibility and adaptability properties, which are desirable in dynamic context aware environments [33]. This is emanated from the fact that these algorithms decide the output or actuation based on hardcoded rules, which cannot be changed dynamically. In this regard, neural networks have been applied to learn from input data to perform decision-making in dynamic settings that require actuations.

Neural network is inspired by human computational activity [35]. It has ability to work in imprecise, uncertain and noisy environment. Neural network can learn, adapt and generalize. It can also classify pattern that has not been before [30].

Neural Networks are made up of several critical components. The largest component is the neuron. A neural network is made up of one or more neurons connected in any configuration. The connecting lines represent weights. Selecting these weights determines how the neural network will respond to particular input patterns. Training neural networks is the method of updating this weight value to decrease the loss (error). Activation functions are used to introduce non-linearity to neural networks. A Sigmoid activation function is widely used function in neural network [37]. Much of neural network has involved using three networks; input layer, hidden layer and output layer. Input layer is the first layer that takes input values and passes them to the next layer. It doesn't apply any operations on the input values. Hidden layer have neurons (nodes) which apply different transformations to the input data. It passes values to the output layer. Output layer is the last layer in the network & receives input from the last hidden layer.

However, neural networks require extensive processing, storage and communication resources to be directly applied in resource-constrained embedded systems because of higher space complexity of the model. This inculcates that there is a need to design a novel lightweight neural network model that can fit to the resource requirements of embedded systems in dynamic context aware environments.

1.2 Motivation

In embedded system developing context aware system is challenging [26][39]. Neural network is ideal solution to solve this challenge. But high memory requirement of the network [37] is recognized and it's understood that NN is not fit to be implemented in resource scared embedded systems. NN consumes higher storage during training [9] and if there is an opportunity in simplifying the training in NN it can fit to be implemented in embedded system so that the problem can be solved. Therefore, the main motivation behind this work is simplifying NN to implement it in embedded system so that it can be context aware.

1.3 Statement of the Problem

In embedded systems, like other IoT applications, the actuation is made based on the information collected from inputs. If the input is not contextualized based on context setting such as the persons, the objects and the computing resources

present in the environment, the system will not respond to a situation in the way it has to respond. Consider a sample scenario of a kitchen where a stove is used in a kitchen for cooking. It is probable that users may leave the stove unplugged in the kitchen while it remained switched on. Furthermore, in a situation where there is intermittent power service, if the power is interrupted while the stove is on, it may remain unnoticed. Apart from damaging the oven due to overheating, the situation can result in causing fire in home or other places. This is a significant potential hazard for damaging properties and human life. This is a compelling evidence for the necessity of developing a context aware embedded system application that switch off the stove by sensing the temperature level in the absence of users or notify users if there is activity of user around the appliance.

The learning system that can contextualize the surrounding environment should have a capability of automatic mechanism of inferring information like humans do [8]. This calls for neural networks that provide an embedded intelligence for smart systems to make decisions at machine speed. However, in current neural networks, input nodes do not compute anything, but simply pass the values to the processing nodes. This can be acceptable for applications that need the same sensors since the degree of inputs have direct proportionality with the output or actuations, but in context aware systems all sensors have no direct relationship with the output. In addition, the increase in weight of some sensors may increase the activation function while others might decrease it.

The main challenge to develop such systems is the constraints in memory size, computational power and other characteristics of embedded systems that can significantly restrict developers from implementing learning algorithms to solve a problem. Furthermore, passing contextualized information from input layer to hidden layer of the neural network might also be another restriction. However, neural networks can be implemented for complex embedded systems if the embedded resources are used efficiently in which all the input are mapped to the knowledge base of neural networks in the internal memory and loaded only once.

Therefore, the main research question for the research is:

- How can we simplify NN to create context aware embedded system?

1.4 Objectives

General Objective

The general objective of this thesis is to design lightweight neural networks so as to show a method for implementing context-aware embedded system in an environment where there is resource limitation.

Specific Objectives

To achieve the general objective, the following tasks are aimed:

- To review literature in the area of neural network and context awareness
- To design lightweight neural network algorithms
- To design testbed using Arduino system
- To implement the prototype of the design algorithms on Arduino
- To train the neural networks by tuning several parameters
- To evaluate the system using sensed data for efficiency and effectiveness

1.5 Methods

The research will begin with a comprehensive review of literatures and industry practices in relation to context aware neural networks in resource-constrained settings to fill a gap in adapting neural networks for embedded systems. Based on the research problems, lightweight neural network algorithms will be designed to be experimented on embedded environments to bring about context awareness.

A testbed will be setup for collecting the data, training and evaluation. In this research, Arduino board will be used as a main experimental device for the proposed algorithms. The algorithms will be implemented/simulated using C++ on Arduino. To implement neural network in embedded systems, different sensors will be used to collect information from the environment. This collected information will be used as an input for the input layer. To train a network, the output value obtained in output layer will be compared with target value to calculate the error. The weight will be updated repeatedly in backward form (starting from output layer through hidden layer to output layer) until the error become equal to the tolerable error (known as threshold error). The trained input and its respective output will be stored in knowledgebase and firing rule will classify every input to one of actuation classes. Efficiency and effectiveness of the algorithm will be evaluated on arduino board by using sensors for collecting input for the test case.

1.6 Scope and Limitations

This thesis aim to provide a solution using neural networks which is limited to modelling complex problem solving numerically [9]. Therefore, this work cannot solve complex problems linguistically using natural language processing.

1.7 Application of Results

This thesis enables to build context aware embedded system by implementing lightweight neural networks. It contributes significantly to the ongoing development of the Internet of Things (IoT) that constitutes smart applications and infrastructures. This has a tremendous application in smart agriculture, eHealth, smart cities and transportations.

1.8 Organization of the rest of the thesis

The rest of this thesis is organized as follows: Section 2 discusses Literature Review; Related work will be discussed in Section 3; Section 4 presents the proposed solution; Section 5 discusses the experimentation and evaluation; and Section 6 presents the conclusion and the future work.

2. Literature Review

2.1 Embedded Systems

Embedded system is a microcontroller based, software driven, reliable, real-time control system [5]. It is designed to perform a specific task. Less requirement of processing power, along with decreasing cost of memory [7] in current technology has led the development of embedded system in a wide range of applications.

The importance of embedded system has been increasing considering application fields where they are used. For example, embedded system has been used in many critical applications such as automotive systems, security systems and home appliances [4]. The use of these applications illustrates the importance of embedded systems.

Our dependence on embedded systems application requires development of new architectural and design techniques [7] in order to meet the necessary performance requirements using their limited resources in terms of processing [39], memory [6], and power [26].

Every embedded system has perceptual faculties through which it can sense aspects of the condition of its environment; it also has actuators that can use to influence the condition of the environment. Complex embedded system applications likely use more than one sensor [13]. In order for the system to take appropriate and smart response to the environmental conditions, the system must be aware of the context of the environment by grouping each sensor into different context.

2.2 Context-aware embedded system

Context awareness is the ability to capture and understand the surrounding contextual information [29]. Therefore context aware embedded system is able to capture context information via embedded devices then takes action without explicit user input [3]. Context aware embedded systems can benefit from machine learning (ML) algorithms potential especially during two distinct phases [15]: the inferences about context information, and the context aware design decisions.

During inference phase ML algorithm is used to find input patterns [19]. During context aware design decisions ML is applied for predicting the output [20], and continuously evolving the adaptation process itself.

2.3 Machine Learning

Machine learning is often applicable in AI as it provides to systems the ability to automatically learn and improve from experience without being explicitly programmed/instructed [16]. The process of learning begins with collecting data based on the application. This is important because the quality and quantity of the collected data determine how good the prediction can be. Ontology, Bayesian network, clustering, decision tree, Markov model, fuzzy logic and neural network are widely used models in machine learning.

Ontology: Ontology is used in Artificial Intelligence to represent a set of concepts within a domain and reason about the properties of that domain [17]. Ontology has components like; individuals, classes, attribute, relationship, function, restrictions, rules, axioms and events. Individual includes concrete object such as people and abstract object such as numbers and words. Classes are category which includes collection of objects. Attributes are used to describe an object. Relationship connects class to object. Function is used to describe complex structures formed in certain relation. Restriction is a description of some assertion to be accepted as true input. Rule is statement that describes the logical inferences which is drawn from assertion. Axioms describe the theory derived from axiomatic statements and an event has contents of the changing of attributes or relations [23].

Ontology can be applied in embedded system design to deal with domain knowledge representation to produce requirement specification with better quality, ambiguity to standardize requirements, inconsistencies for automatic error checking and communication between hardware and software engineers to facilitate communication between engineers by providing common vocabulary among them [21, 23]. Context aware embedded system can also use advantage of solving ambiguity in collecting information from the environment.

Markov Models: The Hidden Markov Model is a statistical based model that is used to analyse and predict current states that involve a series of sequence based on information of previous states. It has been applied in a variety of applications such

as robotics, speech and natural language processing. The model is all about finding the relationships between variables and significance of those relationships [28].

Markov model based algorithms such as Viterbi algorithm is expensive in terms of memory and computational time. For sequence with n length, to find the best path through s states and e edges it requires memory proportional to s^n and time e^n. Forward-backward algorithms are even more expensive [28].

Hidden Markov Model (HMM) can be applied in embedded systems (even though resource constraint is a big issue) to predict actuation based on sensors but predictive accuracy is not their strength [38]. In Markov model prediction is made by splitting the application space to a set of non-overlapping (mutually exclusive) situations [38], which is not natural in context aware system.

Bayesian Network: is a type of probabilistic model that is used to build model from data. Hidden Markov model provides a starting point for dynamic Bayesian network design [31]. Like Markov model, Bayesian network is also statistical based model [21]. BN has a directed acyclic graphs structure. The graph is composed of nodes which represent variables and arcs that represent the relationship between variables. Generally, a BN of n variables consists of a directed graph of n nodes and various arcs. According to the directional graph it is assumed that an event can cause a future event, but not vice versa [17]. This assumption simplifies design of Bayesian network for real time embedded systems when compared to other statistical models (such as hidden Markov Model which is undirected graph).

BN can be applied for embedded systems with smaller input data (in our case scenario - sensors) with less attributes or they end up in over fitting [21]. Bayesian networks have also been applied in context aware system [22] to model users and to predict their needs. Relatively low adaptation effort is needed to apply BN in context aware system with small number of input [23].

Clustering:- is partitioning similar situations into group of situations which are called clusters. The situation is made of all information about the environment that can be collected from sensors [24]. If situations occur over a time is similar to each other then they will be in the same cluster.

Clustering can be applied to context aware embedded systems by grouping information that have the same characteristics on the actuation to the same cluster. In our sample scenario for example, normal hot level of the appliance, existence of a user around the appliance and normal hot level of the appliance, non-existence of the user around the appliance have both the same actuation on the system (Normal state), so these situations can be grouped into the same cluster.

Decision Trees (DT): A decision tree is a type of supervised learning algorithm in which each internal node of a tree represents attribute, each link represents a rule and each leaf represents an outcome class. The general motive of using decision tree is to create a training model which is used to predict class of target variables by learning decision rules based on training data [27]. It tries to solve the problem by using tree representation. Each internal node of the tree corresponds to an attribute, and each leaf node corresponds to a class label. In embedded system design each internal node represents inputs state (in our scenario - sensors) and based on its state the node will be grouped into one of the leaf nodes which corresponds to an actuation class.

Fuzzy Logic: is a computing approach that deals with finding degrees of truth rather than definite true or false (1 or 0). It includes 1 and 0 for extreme true and extreme false respectively, but also handles partial truth between completely true and completely false unlike Boolean logic. The objective of fuzzy logic is to make computers think like human (similar to neural network) and to enable computing with word [12]. The context aware system can make use advantage of the first objective.

Fuzzy logic can be incorporated in embedded systems to enhance performance, increase simplicity, and reduce cost, along with other advantages [25]. It can also be used in context aware embedded system to understand and respond to vague human concepts such as hot, cold, large, small, etc [32].

2.4 Neural Networks

Neural network is one of current learning model that has been implemented in Artificial Intelligence [35], the IoT [16] and other computer science applications [9, 13]. Like fuzzy logic, neural network is also inspired by human computational

activity [33]. Both have ability to work in imprecise, uncertain and noisy environment. But they have also complementary characteristics; fuzzy logic require rule that is set by expert and neural network can learn, adapt and generalize. Neural network can also classify pattern that has not been before [29] which is not true for fuzzy logic. Neural networks are a group of algorithms that have a number of small and simple interconnected components [10]. This networking enables the algorithm to perform more powerful computations by combining the limited processing power of the separate components (or neurons). The algorithm fits perfectly into the multiple-simple-sensors approach, but there is another reason to choose neural networks as well: neural networks are better at tracking record on noisy data than other statistical models [29].

Neural Network basically contains three layers- input layer which takes input signals (values) and passes them on to the next layer, hidden layer which applies different transformations to the input data, and output layer which receives input from the last hidden layer pass the desired values. The output of the i^{th} neuron at the i^{th} layer can be described as:

$$f(x) = \frac{1}{1+e^{-x}} \quad (1)$$

Where *f(x)* is smooth nonlinear function (usually, it is a sigmoid function) and *x* is an output of i^{th} neuron.

Connecting lines between input and hidden layers and between two hidden layers represent weights. Weight is strength of inputs on the output. The weight is randomly created to get the actual output using certain function (such as Eq.1). The weight is updated to reduce the error between actual and target output. The process of updating weight using algorithms (like back propagation) [18] is known as training [10].

Neural networks are good at predicting the outputs between the data points. It is also highly flexible to new tasks and new input patterns and it's good for complex, noisy data from sensor [13]. After weight is iteratively adjusted and error is minimized to threshold, firing technique that classifies the information collected from the environment into an actuation class is required. This firing is used to enable an embedded system to properly act according to the context.

2.5 Firing Rule

A firing rule determines how one calculates whether a neuron should fire for any input pattern. It relates to all the input patterns, not only the ones on which the node was trained. A simple firing rule can be implemented by using Hamming distance technique [16]. The rule goes as follows:

Take a collection of training patterns for a node, some of which cause it to fire (the 1-taught set of patterns) and others which prevent it from doing so (the 0-taught set). Then the patterns not in the collection cause the node to fire if, on comparison, they have more input elements in common with the nearest pattern in the 1-taught set than with the 'nearest' pattern in the 0-taught set. If there is a tie, then the pattern remains in the undefined state [2].

Firing rule algorithm can be used in neural network to get output for untrained inputs. In Hamming distance firing technique, there are only two options one to fire and other not to fire, but in context aware system there should be intermediate outputs between firing and not firing (in our scenario in addition to switching off and not switching off the appliance there should be an alarm as an intermediate output).

3. Related Work

Since the early 90, ML has been applied to support different frontage of context aware systems [12]. However, this work is dedicated to explore distinct applications of neural networks for context-aware embedded systems.

Bayesian network to model and reason about the uncertain contexts was received much attention by the context-aware research community [17]. Gu *et al.* [30] represented a particular Bayesian network in the Ontology Web Language (OWL) based ontology and then translated into a Bayesian network for reasoning. They propose a probability extension to an ontology-based model for representing uncertain contexts; and use Bayesian networks to reason about uncertainty. In addition, they have incorporated the supports of probabilistic markups and Bayesian networks into context-aware middleware system to enable the building of context-aware. They have represented contexts as first-order predicate calculus and described structures and properties of context predicates in ontology. They show Bayesian network is a powerful enough for reasoning about causal relationships between various uncertain contexts.

However, such approaches have imperfect property about uncertain result of context awareness problem and none of them consider the issue of reusing the uncertain knowledge captured by Bayesian networks even though ontology was used. The knowledge captured by a particular Bayesian network is fixed and distinctive for a particular application so that it is unable to share and reuse between applications. According to the authors, relatively low adaptation effort is needed to apply BN in embedded system with small number of input, but for the embedded system to react properly according to the context a number of inputs (usually sensors)are required.

Yavari [1] proposed a means to transform context obtained from environment to develop embedded system-enabled awareness services by storing contextualized data in separate server. The author proposed an approach and techniques for performing internet-scale data contextualisation. In particular, IoT-based contextualisation techniques were used to collect data in cities and use such data to provide information that best suits the context of each user. The author exemplifies the proposed contextualisation solution in a smart parking space recommender

service. They have evaluated if contextualisation helps process driver queries faster and can handle Internet-scale IoT data sets. They achieve better contextualisation performs when the amount of data increases towards the Internet scale.

But the system must always communicate with the server before making action based on situation. This adds another task to the system and it's difficult to provide real-time output. Moreover, functionality of the system is dependent on functionality of the server and it cannot provide contextualised decision when there is a failure in server.

Faridi [28] have applied HMM to contextualize activities in context aware system. The author proposed the use of HMM to predict and infer the context of the user based on the location of the target, the activity user is doing, identity of the target and the time in which the activity occur. The author has shown that by using HMM it is possible to predict a user's next state given the current state and previous states.

Although HMMs are recognized as one effective technique for activity classification, because they offer dynamic time warping, have clear Bayesian semantics and well-understood training algorithms, its computational expense (both memory and time complexity) and floating-point representation of the parameters makes it very prone to numerically underflow. This makes it less likely feasible to implement it in context aware embedded system for two main reasons. First, such an algorithm with high computational expense is not fit to be implemented in resource limited embedded device. Second, usually many activities are performed in a dynamic environment and some of the activities are short lived. The time of switching some data might be lost and handling of floating numbers in such a dynamic environment has an effect on smart prediction of the output. The authors have substituted floating-point operation by integer operation for parameter representation to implement forward algorithm for motion classification on embedded system. However, it still causes intermediate results to overflow. The running time of forward algorithm is $O(K^2N)$ where, K is number of states and N is number of observation. This time complexity is huge for embedded system.

Daniel [31] presented an approach of modelling contextual information of a context-aware system using the example of a 'context-aware in-car infotainment system'. In particular, the author showed how developers of context-aware in-car

infotainment systems can model reliability calculations of contextual information and handling of multiple sources of contextual information by using ontology-based modelling technique. It allows complex reasoning and presentation with more meaningful results. Its main weakness is that data should be in OWL file format and also this method has low performance.

Ning [32] proposed middleware architecture for context processing in IoT. The architecture is based on fuzzy logic control (FLC) system for context reasoning. The author proposed a formal context representation model in which a user's context is described by a set of roles and relations correspond to a context space.

The algorithm is simple, easy to extend, requires less resources and represented in natural language. However, the algorithm requires knowledge of human expert that mimic his/her thinking and as a result it is prone to have manual entry mistakes from the expert that uses the algorithm. But context aware embedded systems can take advantage of fuzzy logic if it is combined with neural network to process linguistic information along with other benefits.

Hongyuan W. [33] applied rule induction for context aware adaptation by taking emergency situation as a scenario. According to the author, the "rule has respective meaning for system, software and context".

"Rules" usually take the form:

Condition -> Action

"Adaptation" rules take the form:

Contextual condition -> Software system action

Rules for "Context" state change would be:

Condition and action -> Context state after change

The algorithm is the simplest and straight forward with less resource requirements when compared to HMM, BN and NN. But mistakes can be made during writing the rule and wrong rule will result in bad output prediction.

Another model which can be applied in context aware embedded system development is neural network that have rapidly gained popularity for it success in image recognition, natural language processing, and other application areas [4, 35, 36].

Bashyal *et al.* [13] created an embedded neural network for fire classification. It has seven inputs and three outputs. Each of the seven inputs is a homogeneous sensor and the three outputs represent the three types of fires to be classified: No

Fire, Class A, and Class B, where class A is high and class B is low fire detected. This network is very large relative to the simplicity of the problem to be solved. It could have been done using simple logic.

It is not clear to know whether the entire network is working or a small number of neurons are carrying the load. It is also good to notice that the numbers of layers and nodes affect the time complexity of the use of a neural network.

Another reason why additional hidden layers seemed to be a problem was that they would require a very extensive training set to be able to compute weights for the network. Unlike specific applications, many different sensors are required for context aware systems. So, we need to have optimized technique so that we only use one sensor once. For example, Bashyal *et al*. [13] have used a homogeneous gas sensors seven times; this is not possible for context aware systems which require various heterogeneous sensors to collect information from an environment. In addition when multi sensors are used some sensors can increase the activation and others may decrease it. This cannot be considered when similar sensors are used and the case has not also been taken into account in other neural network applications that use sensors.

Based on the analysis of the related works presented above, we notice that embedded systems can use advantage of machine learning models ability of learning by example, self-learning and self-organizing based on example and processing logical information to be a context aware system. We also notice that computational expense is an issue in adapting machine learning models to embedded system. In addition it is also noticed that there is a gap in using efficient power of neural networks in adapting it in complex embedded system applications that require smart decisions. This thesis focuses on filling the gap in using heterogeneous input sensors and deploying neural networks on resource limited embedded devices.

4. The Proposed Solution

4.1 The Proposed Architecture

In the domain of context awareness, a technology cannot be described as context aware without at least exhibiting the ability to detect a particular situation such as activity, condition and effect of the situation on the system. Having the required elements available, the next step would be to apply and utilize inference technique to correctly associate the combination of perceived situation to the correct context. In our case, we utilized neural network at the heart of embedded systems (as shown in Figure 4.1) as our inference techniques of choice. We use scenario of stove controller which is one of the widely used home appliances.

In our deign of neural network for context aware embedded system, there are input elements (in our case sensors) that have direct relationship with the output (actuation) and some of them have reverse or indirect relationship. In addition to weight of input element in input layer of a neural network, state of each input elements is passed from input layer to hidden layers. In this work, the state of input elements are classified into three; input element that has direct relationship with the output has state 1, input element that has indirect relationship with an output has state -1, and the one that is not active or has no effect on the output has state 0.

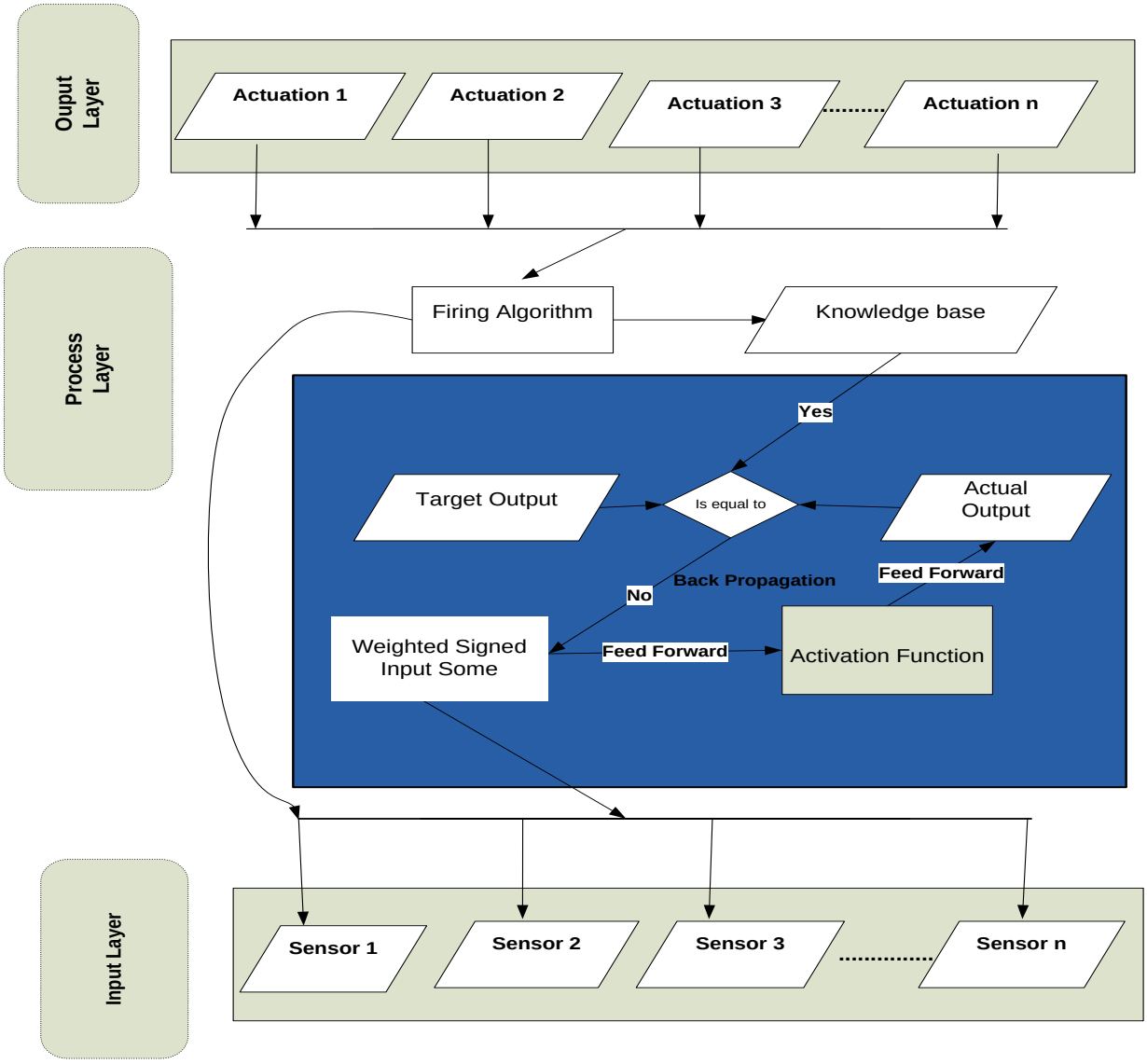

Figure 4.1: Layered architecture for neural network adaptation for embedded system

In Figure 4.1 input from different sensors will be passed to the hidden layer of the neural network. To calculate the input value; state, input value and neuron weight of each sensor will be multiplied and the sum of the result will be passed to the activation function. In this work, weight of neuron in the input layer and hidden layer is assigned by using Guassian distribution [14]. The difference between target output and actual output (the output that is calculated by activation function) is calculated to find error during training the neural network. The trainings repeated by updating weight of the network until the error becomes equal to threshold error. When the total error between target and actual output becomes less than or equal to the threshold error, the output for corresponding inputs will be stored in a

knowledgebase. Based on values in the knowledgebase the firing algorithm is reacted to the environment using information from sensors. In our design it is good to notice that training is done on high capacity computer before deploying the system on embedded device; this will make the NN to fit for resource limited embedded device. Once it is deployed on embedded device the firing algorithm is used to react to the environment according to the context by using knowledgebase; No training needed.

The NN that is used in this work is different from other NN design for two main reasons. First, after the system is deployed on embedded device, the actuation is made without training the network; this reduces computational expense (both memory and time). The second reason is the firing algorithm that classifies all inputs (including the untrained inputs) to actuation class is used in this design; this is used to make the embedded system to be flexible in dynamic environments.

In our scenario increase in weight of input from temperature sensor has direct relationship with actuation which switches off the appliance or alerts the user, whereas input from motion sensor (which senses presence of a user) has indirect proportionality with the actuation. Change in load of the appliance on stove has also directly proportional to the actuation and it is calculated as:

$$\Delta Wt = wt_i - wt_c \quad (2)$$

where, ΔWt is change in load, wt_i is initial load and wt_c is current load.

If $\Delta W>0$, the state will be 1, otherwise the state will be 0.

Neural network design for the scenario is shown in Figure 4.2.

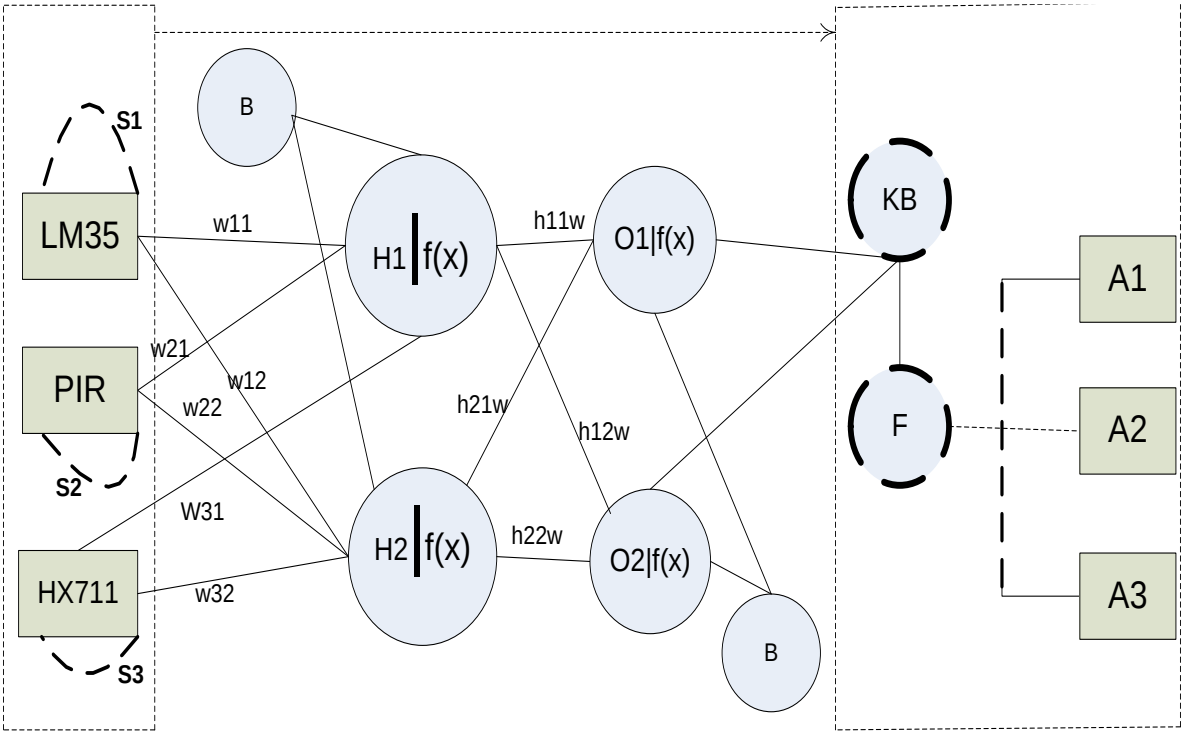

Figure 4.2: Context aware embedded system for stove controller using neural networks

In Figure 4.2 above, LM35, PIR and HX711are temperature, motion and load sensors respectively. S1, S2, and S3 are sensor states (-1, 0 or 1), H1 and H2 are hidden nodes, w11, w12, w21, w22, w31, and w32 are network weights of respective inputs, h11w, h12w, h21w, and h22w are weights of hidden layers, O1 and O2 are output nodes. B is bias and its value is always one, f(x) is activation function (we use sigmoid function), A1, A2 and A3 are actuation classes i.e. A1 switching off, A2 alert, A3 normal, KB is knowledgebase which store output for trained inputs. F is firing rule algorithm which classify the contextual information into actuation class based on the knowledgebase.

On Figure 4.2, broken lines represent design used specifically in this work and others are taken from neural network design [37]. In our scenario we use three sensors as input, two hidden nodes, two output nodes and three actuation classes. But our design is not limited to these amount of nodes, a number of nodes can be used in each layer as required based on the application; Of course computational power should be considered specially for the number of nodes in hidden layers.

4.2 Firing Rule for the proposed design

Firing rule algorithm used in this paper is different from other technique such as Hamming distance technique (section 2.5) in firing options. In our technique, in addition to firing and not to fire, how far the input is from firing and not firing should be known and based on the distance the output can be classified into one of the intermediate classes. The rule in this technique goes as follows:

Take a collection of patterns for a node, some of which cause it to fire (the 1-taught set of patterns) and others which prevent it from doing so (the 0-taught set). Input patterns not in both collections cause the node to fire in intermediate actuation classes. The intermediate class to be fired is known by N-m where N is total number of classes and m is number of similar values in a pattern of the nearest class.

identify 1-taught set pattern //firing pattern
identify 0-taught set pattern //not firing pattern
get patterns from a node
if (pattern = 1-taught set)
then
fire the first_actuation_class
else if (pattern = 0-taught set)
then
fire the last_actuation_class
else
determine the intermediate class
1-taught nearest pattern = count similar values in 1-taught pattern and current pattern
0-taught nearest pattern = count similar values in 0-taught pattern and current pattern
if (1-taught nearest pattern = 0-taught nearest pattern)
then
intermediate class = Round(total number of classes/2,0)
else if (1-taught nearest pattern > 0-taught nearest pattern)
then

intermediated class = (total number of classes)–(1-taught nearest pattern)

else

intermediated class = (total number of classes)–(0-taught nearest pattern)

fireintermediate_class

***Algorithm 4.1**: Firing rule algorithm for the proposed model*

For example, in Table 4.1 a 3-input neuron is taught to output A1 when the input (X1, X2 and X3) is 111 and A3 when the input is 000. Then, By applying the firing algorithm in every column the following truth table is obtained for three actuation classes A1, A2, A3;

***Table 4.1**: Sample firing rule technique for the proposed solution*

X1	**X2**	**X3**	**Class**
0	0	0	A3
0	0	1	A2
0	1	0	A2
0	1	1	A2
1	0	0	A2
1	0	1	A2
1	1	0	A2
1	1	1	A1

In Table 4.1 the output is known only for two input patterns i.e 111 and 000. The output for the rest of input patterns can be found based on the two input patterns knowledge. NN design in this work can take advantage of this algorithm to react to dynamic environment that has not been trained.

As an example of the way the firing rule is applied, take the pattern 101. It differs from 111 in 1 element and from 000 in 2 elements. Therefore, the pattern is nearest

to111 patterns. To determine firing class next to either A1 or A3, we use the formula N-m. In our case we have three classes and N is three, we have two similar patterns in classes where it belongs therefore m is two. So intermediate firing class will be the first (3-2) class next to A1 which means class A2 will be fired. The same is true for other patterns.

4.3 Training

Training a neural network basically means adjusting all of the weights by repeating two key steps, forward propagation and back propagation. Figure 4.3 [18] shows how forward and back propagations work.

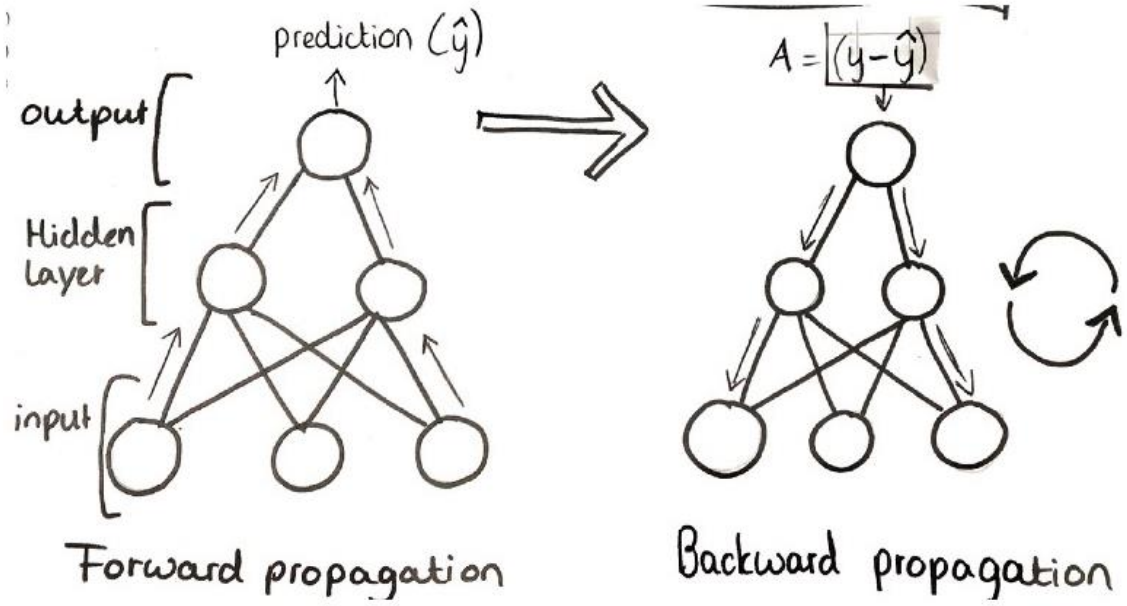

Figure 4.3: Overview of forward and backward propagation [18]

4.3.1 Forward Propagation

In forward propagation, we apply a set of weights to the input data and calculate an output. For the first forward propagation, the set of weights is assigned by Guassian distribution.

Value from sensors is input for the hidden layers and outputs from hidden layers are an input for output node/s. For the sensors in addition to input value, the effect of the sensor should also be known. To do these four things are done here as described in Algorithm 4.1.

First input from each sensor is multiplied by a state as follows:-

$$Ii=Xi.Si \quad (3)$$

where, Xi = x1, x2, x3....,xn Si = s1, s2, s3..., sn

where, Xi are inputs, Si are state of respective inputs and Ii is signed inputs.

In our scenario the signed inputs are computed as follows:-

I1=LM35.s1, I2= PIR.s2 and I3= HX711.s3,

Next, all signed inputs are multiplied by a weight

$$I1 \rightarrow I1*w11, I1*w12 \quad (4)$$

$$I2 \rightarrow I2*w21, I2*w22 \quad (5)$$

$$I3 \rightarrow I3*w31, I3*w32 \quad (6)$$

Third, all weighted signed inputs for each hidden nodes are added together with a bias.

$$[Hi] = Ii.[W] + B \quad (7)$$

where Hi = H1, H2, H3..., W= w11,w12, w21, w22....

$$[H1, H2] = [I1 \quad I2 \quad I3]. \begin{bmatrix} w11 & w12 \\ w21 & w22 \\ w31 & w32 \end{bmatrix} + B$$

H1 = (I1*w11 + I2*w21 + I3*w31) + B

H2 = (I1*w12 + I2*w22 + I3*w32) + B

Finally, the sum is passed to each hidden nodes through activation function.

$$y1 = f(H1) \quad (8)$$

$$y2 = f(H2) \quad (9)$$

where, y1 and y2 are outputs from hidden layers.

y1 and y2 are outputs from hidden layers (H1 and H2 respectively) and next it will be input for output node. Three things are done to compute output for the output node.

First each value from hidden layers are multiplied by a weight

y1 → y1*h11w, y1*h21w

y2 → y2*h12w, y1*h22w

Next, all the weighted inputs are added together with a bias:

O1 = (y1*h11w + y2*h21w) + B and O2 = (y1*h12w + y2*h22w) + B

And again the sum is passed through an activation function:

$y_{O1=}f(y1*h11w + y2*h21w + B)$

y_{O2}=f(y1*h12w + y2*h22w + B)

Where, y_O output from output nodes.

The activation function is used to turn an unbounded input into an output that has a predictable form. Throughout this work we will use sigmoid activation function. The formula for a sigmoid function is [15]:

$$sigmoid(z) = \frac{1}{1 + e^{-z}} \tag{11}$$

Where, z is an output of hidden nodes and output nodes. h1's output: z = I1*w11 + I2*w21 + B =H1, h2's output: z = (I1*w12 + I2*w22) + B, O1's output:y1*h11w + y2*h21w + B and O2's output: y1*h12w + y2*h22w + B.

The sigmoid function only outputs numbers in the range (0,1) as shown on Figure 4.4 [18]. If z is very large, e^{-z} will be close to 0, and therefore the output of the sigmoid will be 1. Similarly, if z is very small, e^{-z} will be infinity and hence the output of the sigmoid will be 0.

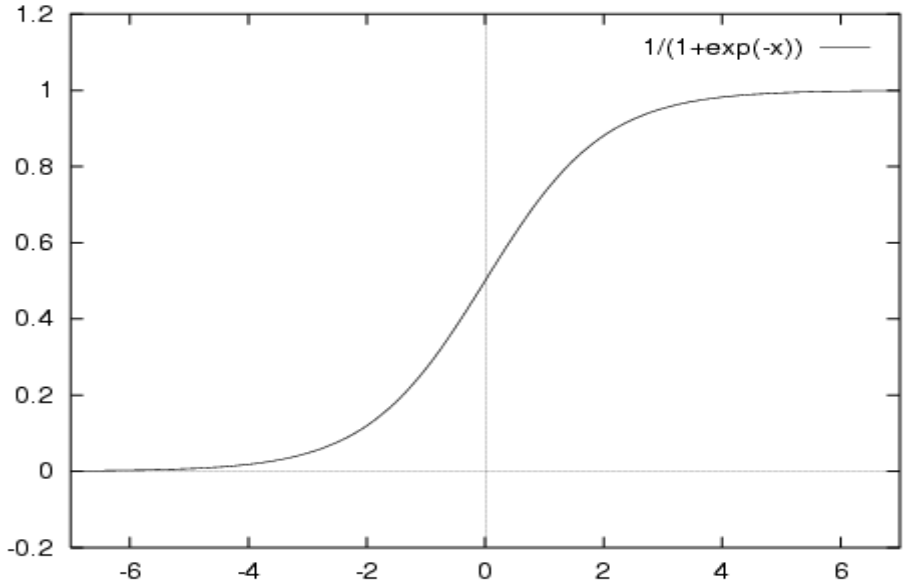

Figure 4.4: Sigmoid function graph [18]

Algorithm 4.2 describes forward propagation algorithm.

// Getting an input

Get input for all input nodes

Assign state to all input nodes

Multiply state with input values for respective input nodes to get signed input value

Initialize all weights by using guassian distribution

repeat

// Propagated the input forward through the network

for every network first hidden layer

for every node in the first hidden layer

Multiply signed input with respective weight
Calculate weight sum of the inputs and bias to the node
Calculate activation function for the node

end

end

for each hidden layer after first hidden layer and output layer in the network
for every node in the layer
Calculate the weight sum of the inputs to the node
Calculate the activation for the node to get actual output
end
end

// making a prediction

if value in output layer neuron is equal to target output

then, apply firing rule technique

classify the output into one of actuation class to make decision

else

// Propagated the output backward through the network

back pass the output

while(maximum number of iterations < than specified)

***Algorithm 4.2**: Forward propagation for the proposed solution*

4.3.2 Backward Propagation

Our work is different from others in forward propagation and the backward propagation is adapted from [18]. In back propagation, we measure the margin of error of the output and adjust the weights accordingly to decrease the error. Error is first calculated by computing the difference between actual outputs which is forward propagation and the target output which is true output. Algorithm 4.2 describes how this can be done.

Error in each output nodes is calculated by mean square error (MSE) as follows [11]

$$E_{Oi} = \frac{1}{2}(\text{target output} - \text{actual output})^2 \quad (12)$$

Total error can be computed as follows:-

$$E_{total} = \sum \frac{1}{2}(\text{target output} - \text{actual output})^2$$

Our main goal of the training is to reduce the total error or the difference between actual output and target output. Since target output is constant, the only way to reduce the error is to change the variable elements affecting actual value which is weight. Gradient descent is used in back propagation to update weight. It is an iterative optimization algorithm for finding the minimum error function.

To decrease total error partial derivatives of error with respect to weight is computed as follows [2];

$$\frac{\partial E}{\partial w^k ij}, \quad (13)$$

Where, E is total error and w is weight in k layer.

The derivation of the error function is evaluated by applying the chain rule as following;

$$\frac{\partial E}{\partial w^k ij} = \frac{\partial E}{\partial yoi^k} * \frac{\partial yoi^k}{\partial w^k ij}$$

Where yo is an actual output in k layer

This can be further decomposed as follows

$$\frac{\partial E}{\partial w^k ij} = \frac{\partial \sum \frac{1}{2}(target\ output - actual\ output)^2}{\partial yoi^k} * \frac{\partial yoi^k}{\partial w^k ij}$$

Computation of the error terms will proceed backwards from the output layer down to the input layer.

a. Output layer

We want to know how much a change in w_{ij} affects the total error $\frac{\partial E}{\partial w^k ij}$ using a chain rule.

$$\frac{\partial E}{\partial w^k ij} = \frac{\partial E}{\partial yoi^k} * \frac{\partial yoi^k}{\partial netoi} * \frac{\partial netoi}{\partial w^k ij}$$

Next, how much does the output of o_i change with respect to its total net input;

The partial derivative of the sigmoid function is the output multiplied by 1 minus the output [2]:

$yo_i = \frac{1}{1+e^{-netoi}}$, where o_i is net input which is weighted sum

$$\frac{\partial yoi}{\partial oi} = yo_i(1 - yo_i) \qquad (14)$$

Finally, how much does the total net input of o_i change with respect to w_{ij};

It is computed as-

$$\frac{\partial netoi}{\partial wij} \qquad (15)$$

When we put them together using equation 5 we can calculate total error with respect to all weights.

Alternatively, we can use delta rule to calculate $\frac{\partial E}{\partial netoi}$ as follows and we can use this to rewrite the calculation above [2]:

$$d_{oi} = \frac{\partial E}{\partial yoi} * \frac{\partial yoi}{\partial netoi} = \frac{\partial E}{\partial netoi} \qquad (16)$$

where, *d* is delta then,

$d_{oi} = -(target_{oi} - actual_{oi}) * actual_{oi}(1 - actual_{oi})$

Therefore:

$$\frac{\partial E}{\partial w^k ij} = d_{oi}.actual hik^{-1}$$

Where k is network layer

To decrease the error, we then subtract this value from the current weight as:

New weight = old weight - $\frac{\partial \mathrm{E}}{\partial \mathrm{w}^{k}ij}$

New wij= wij - $\frac{\partial \mathrm{E}}{\partial \mathrm{w}^{k}ij}$

Optionally the delta is multiplied by some learning rate [36] , which we'll set to 0.5:

$$\mathrm{wij} = \mathrm{wij} - \Theta.\frac{\partial \mathrm{E}}{\partial \mathrm{w}^{k}ij} \qquad (17)$$

Learning rate is mainly used for controlling how much we are adjusting the weights of our network with respect to the loss gradient.

We perform the actual updates in the neural network after we have the new weights for all weights in all layers (ie, we use the original weights, not the updated weights, when we continue the back propagation in hidden layer).

b. Hidden Layer

Next, we'll continue the backwards pass by calculating new values for weights leading into the next hidden layer neurons.

This can be done as follows:

$$\frac{\partial \mathrm{E}}{\partial \mathrm{w}^{k}ij} = \frac{\partial \mathrm{E}}{\partial \mathrm{ah1}} * \frac{\partial \mathrm{aHi}}{\partial \mathrm{netHi}} * \frac{\partial \mathrm{netHi}}{\partial \mathrm{w}^{k}ij}$$

where, aHi is actual output in for hidden layer neuron i, netHi is net input to hidden neuron which is weighted sum leading into the neuron.

We use a similar process as the output layer, it only differ in that the output of each hidden layer neuron contributes to the output of multiple output neurons. For example aHi affects all y_{oi}. Therefore the $\frac{\partial \mathrm{E}}{\partial \mathrm{aHi}}$needs to take into consideration its effect on the output neurons:

$$\frac{\partial \mathrm{E}}{\partial \mathrm{aHi}} = \frac{\sum \partial \mathrm{Eyoi}}{\partial \mathrm{aHi}}$$

where, Ey_{oi} is error in each output neurons

$$\frac{\partial \mathrm{Eyoi}}{\partial \mathrm{yoi}} = \frac{\partial \mathrm{Eyoi}}{\partial \mathrm{netoi}} * \frac{\partial \mathrm{netoi}}{\partial \mathrm{yoi}}$$

We can calculate $\frac{\partial \mathrm{Eyoi}}{\partial \mathrm{netoi}}$as:

$$\frac{\partial \mathrm{Eyoi}}{\partial \mathrm{netoi}} = \frac{\partial \mathrm{Eyoi}}{\partial \mathrm{yoi}} * \frac{\partial \mathrm{yoi}}{\partial \mathrm{netoi}}$$

And $\frac{\partial \mathrm{netoi.}}{\partial \mathrm{yoi}}$is known.

Therefore:

$\frac{\partial \mathrm{E}}{\partial \mathrm{aHi}} = \sum \frac{\partial \mathrm{Eyoi}}{\partial \mathrm{aHi}}$

Now that we have ,$\frac{\partial \mathrm{E}}{\partial \mathrm{aHi}}$we need to figure out $\frac{\partial \mathrm{aHi}}{\partial \mathrm{netHi}}$and then $\frac{\partial \mathrm{netHi}}{\partial \mathrm{wij}}$for each weight:

$$\mathrm{aHi} = \frac{1}{1 + e^{-netHi}}$$

$$\frac{\partial \mathrm{aHi}}{\partial \mathrm{netHi}} = aHi(1 - aHi)$$

We calculate the partial derivative of the total net input to Hi with respect to wij the same as we did for the output neuron:

$$\frac{\partial \mathrm{netHi}}{\partial \mathrm{wij}} = weighted\ sum\ leading\ into\ Hi$$

Finally total error with respect to weight is computed as:

$$\frac{\partial \mathrm{E}}{\partial \mathrm{wij}} = \frac{\partial \mathrm{E}}{\partial \mathrm{aHi}} * \frac{\partial \mathrm{aHi}}{\partial \mathrm{netHi}} * \frac{\partial \mathrm{netHi}}{\partial \mathrm{wij}}$$

We can now update wij:

$\mathrm{wij} = \mathrm{wij} - \theta.\frac{\partial \mathrm{E}}{\partial \mathrm{wij}}$

Algorithm 4.3 describes backward propagation.

```
Assign all network inputs and output

Initialize all weights with Guassian distribution method

repeat

   for every pattern in the training set

         Present the pattern to the network

//       Propagate the errors backward through the network
          for every node in the output layer
               Calculate the error signal

                    Update each node's weight in the network
                    end

          for all hidden layers
                    for every node in the layer
                              Calculate the node's signal error
                              Update each node's weight in the network
                    end
          end

//       Calculate Global Error
          Calculate the Error Function

//       Propagated the input forward through the network:
                    for each layer in the network
                              for every node in the layer
                                        Calculate the weight sum of the inputs to the node
                                        Add the threshold to the sum
                                        Calculate the activation for the node
                              end
                    end

          end

while ((maximum  number of iterations < than specified) AND
       (Error Function is > than specified))
```

***Algorithm 4.3**: Backward Propagation*

Neural networks repeat both forward and back propagation until the weights are calibrated to accurately predict an output.

4.4 Evaluation of the Scenario

In training the network inputs are first feed by forward propagation. In addition to the input values, state of an input will be feed then weight is assigned to an input. We assign weight by using Gaussian distribution rather than random assignment as shown on Figure 4.5. This can minimize iterations for updating weight (therefore optimize resource constraints) than random assignment of weight.

For example, if 2, 5, 9, 1 are assigned for w11, w12, w21, w22 respectively, then by using Guassian distribution the value for w11 will be$\frac{2-mean}{stdv}$, where stdv is standard deviation. stdv=3.11 and mean is 4.25. Thus w11 = -2.25/3.11= -0.72.

Similarly, w12= 0.24, w21= 1.52 and w22 = -1.045. Let us assign state -1 for S1 and state 1 for S2. The value for bias in all neuron is 1.

If weight 4, 1, 3, 2 are assigned for h11w, h12w, h21w, h22w. By applying similar method in hidden layer, h11w = 1.08, h12w = -1.08, h21w = 0.36, h22w = -0.36.

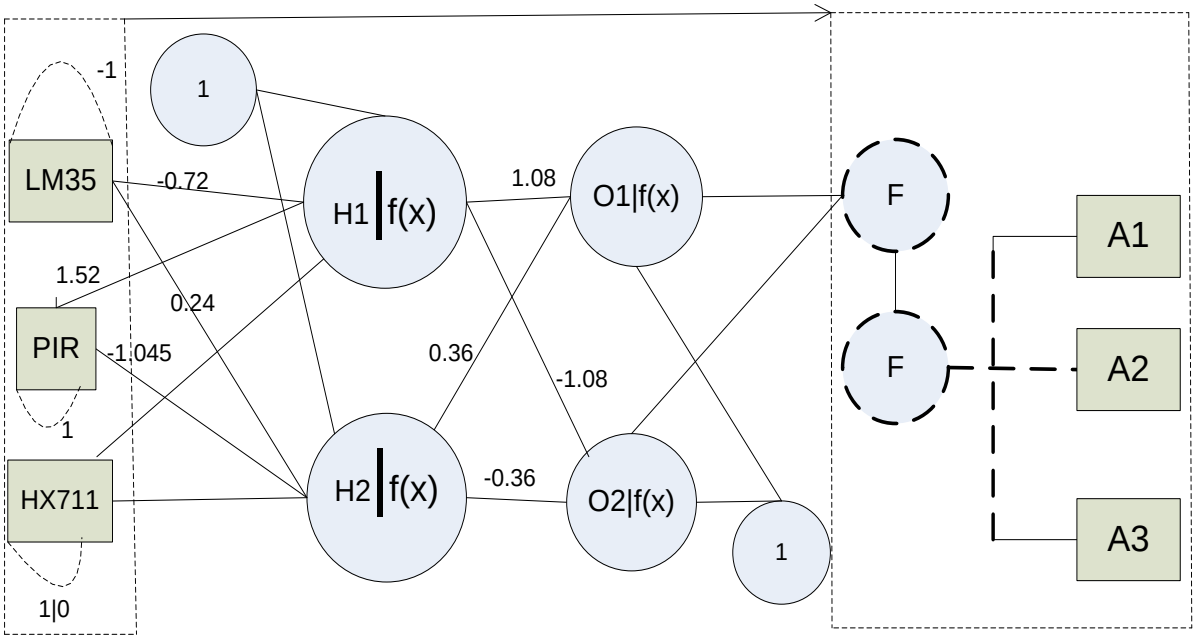

Figure 4.5: Feeding inputs, assigning state to inputs, assigning weight to inputs and hidden neurons for the scenario

Now we feed input to two neurons (in our case LM35, PIR) in input layer (we skip HX711 for simplicity) then give target outputs for all input patterns as Table 4.2;

Table 4.2: *Target outputs for all input patterns in the case scenario*

A	B	O1	O2
0	0	0	0
0	1	1	1
1	0	0	0
1	1	0	1

State of an input is multiplied by the input value. For example if we take 01 pattern for inputs, the target output will be 11.

To get the actual output five things are done:

First, 0 is multiplied by -1 to become 0 and 1 is multiplied by 1 to become 1 therefore the input now becomes 01 it's not always the same for all input patterns sometimes sign can be changed based on the state.

Next, the signed input from neurons is multiplied by respective weights to get net input for hidden neuron:

$netH_1 = I_A*w11 + I_B*w21 + B$, $_{net}H_1 = 0* -0.72 + 1*1.52 + 1 = 2.52$

$netH_2 = I_A*w12 + I_B*w22 + B$, $_{net}H_2 = 0*0.24 + 1*-1.045 + 1 = -0.045$

Third, actual output for hidden neuron is then calculated by sigmoid activation function:

$aH_1 = \frac{1}{1+e^{-netH1}} = \frac{1}{1+e^{-2.52}} = 0.93$

$aH_2 = \frac{1}{1+e^{-netH2}} = \frac{1}{1+e^{-(-0.045)}} = 0.49$

This output is used as an input for neuron in output layer.

Fourth, net input for output neuron is calculated as:

$netO_1 = aH_1*h11w + aH_2*h21w + B$, 0.93*1.08 + 0.49*0.36 +1 = 2.18

$netO_2 = aH1*h12w + aH_2*h22w + B$, 0.93*-1.08 + 0.49*-0.36 +1 = -0.18

Finally, actual output for output layer neuron is calculated by sigmoid activation function:

$yo_1 = \frac{1}{1+e^{-netO1}} = \frac{1}{1+e^{-2.18}} = 0.898$

$yo_2 = \frac{1}{1+e^{-netO2}} = \frac{1}{1+e^{0.18}} = 0.457$

The actual output is 0.898 and 0.457 where the target is 1 and 1 respectively. As we can observe, there is an error in both outputs. This error should be or close to zero and this will be done by back propagation.

Calculating the Total Error

We can now calculate the error for each output neuron:

$E_{total}=\sum\frac{1}{2}(t_o - y_o)^2$ where, t_o is target output and y_o is actual output.

$Eo_1= \frac{1}{2}(t_{o1} - y_{o1})^2 = \frac{1}{2}(1 - 0.898)^2 = 0.0052$

$Eo_2 = \frac{1}{2}(t_{o2} - y_{o2})^2 = \frac{1}{2}(1 - 0.457)^2 = 0.1474$

Therefore, $E_{total} = E_{o1} + E_{o2} = 0.1526$

Backward propagation

Our goal in back propagation is to adjust each of the weights in the network so that they cause the actual output to be closer to the target output. The weight is updated in backward (first weights in output layer is updated then weights that goes to hidden layers are updated).

Weight in output layer

Consider . H11w We want to know how much a change in H11w affects the total error,

$\frac{\partial E}{\partial H11w}$. It can be calculated by delta rule:

$\frac{\partial E}{\partial H11w}$ = -(target$_{o1}$ – actual$_{o1}$)*actual$_{o1}$(1 - actual$_{o1}$)*aH1.

$d_{o1} = \frac{\partial E}{\partial actualO1} * \frac{\partial actualO1}{\partial netO1} = \frac{\partial E}{\partial netO1}$

d_{o1} = -(target$_{o1}$ – actual$_{o1}$)*actual$_{o1}$(1 - actual$_{o1}$)

Therefore:

$$\frac{\partial E}{\partial H11w} = -do1 * aH1$$

$\frac{\partial E}{\partial H11w} = -(-(1 - 0.898)* 0.898(1 – 0.898))*0.93 = 0.0087$

To decrease the error we subtract this value from weight:

H11w = H11w- $\theta.\frac{\partial E}{\partial H11w}$ where θ is learning rate and its value is 0.5.

H11w = 1.08 – 0.5*0.0087 = 1.07. Therefore new weight for H11w is 1.07

Similarly,

$$\frac{\partial E}{\partial H21w} = -do1 * aH2$$

$\frac{\partial E}{\partial H21w} = -(-(1 - 0.898)* 0.898(1 – 0.898))*0.49 = 0.0046$

H21w = H21w- $\theta.\frac{\partial E}{\partial H21w}$= 0.36 – 0.0023 = 0.35

$$\frac{\partial E}{\partial H12w} = -do2 * aH1$$

d_{o2}= -(target$_{o2}$ – actual$_{o2}$)*actual$_{o2}$(1 - actual$_{o2}$)*aH1

*=-(1-0.457)*0.457(1-0.457)*0.93= -0.125*

H12w = H12w – $\theta.\frac{\partial E}{\partial H12w}$ = -1.08 -0.0625 = -1.02

$$\frac{\partial E}{\partial H22w} = -do2 * aH2$$

d_{o2}= -(target$_{o2}$ – actual$_{o2}$)*actual$_{o2}$(1 - actual$_{o2}$)*aH2

*=-(1-0.457)*0.457(1-0.457)*0.49= -0.066*

H22w = H22w – $\theta.\frac{\partial E}{\partial H22w}$ = -0.36 -0.033 = -0.32

Weight in hidden layer

Next, we calculate new values for w11, w12, w21 and w22. When we start with w11:

$$\frac{\partial E}{\partial w11} = \frac{\partial E}{\partial aH1} * \frac{\partial aH1}{\partial netH1} * \frac{\partial netH1}{\partial w11}$$

$$\frac{\partial E}{\partial aH1} = \frac{\partial Eo1}{\partial aH1} + \frac{\partial Eo2}{\partial aH1}$$

$$\frac{\partial Eo1}{\partial aH1} = \frac{\partial Eo1}{\partial netO1} * \frac{\partial netO1}{\partial aH1}$$

$$\frac{\partial Eo1}{\partial netO1} = \frac{\partial Eo1}{\partial yo1} * \frac{\partial yo1}{\partial netO1}$$

$\frac{\partial Eo1}{\partial yo1} = -0.102, \frac{\partial yo1}{\partial netO1} = 0.091$

$$\frac{\partial Eo1}{\partial netO1} = 0.009$$

$$\frac{\partial netO1}{\partial aH1} = 1.08 * aH1 + 0.36 * aH2 = 1.08 * 0.93 + 0.36 * 0.49 = 1.18$$

$$\boldsymbol{\frac{\partial Eo1}{\partial aH1}} = 0.009 * 1.18 = 0.0106272$$

$\frac{\partial Eo2}{\partial netO2} = 0.248151 * 0.543 = 0.134745993$

$\frac{\partial netO2}{\partial aH1} = -1.08 * 0.93 + -0.36 * 0.49 = -1.1808$

$\boldsymbol{\frac{\partial Eo2}{\partial aH1}} = 0.134745993 * -1.1808 = 0.15910806853$

Therefore,

$$\boldsymbol{\frac{\partial E}{\partial aH1}} = 0.0106272 + 0.15910806853 = 0.1697352685$$

$\frac{\partial aH1}{\partial netH1} = aH1(1 - aH1) = 0.93(1 - 0.93) = 0.651$

$netH1 = 0 * w11 + 1.52 * w21 + 1$

$\frac{\partial netH1}{\partial w11} = I_A = 0$

$\frac{\partial E}{\partial w11} = 0.1697352685 * 0.651 * 0 = 0.$

This means the old and new weight for w11 is the same.

For weight w12:

$\frac{\partial E}{\partial w12} = \frac{\partial E}{\partial aH2} * \frac{\partial aH2}{\partial netH2} * \frac{\partial netH2}{\partial w11} * 0.2499 * 1$

$\frac{\partial E}{\partial aH2} = \frac{\partial Eo1}{\partial aH2} + \frac{\partial Eo2}{\partial aH2}$

$$\frac{\partial Eo1}{\partial aH2} = \frac{\partial Eo1}{\partial netO1} * \frac{\partial netO1}{\partial aH2}$$

$\frac{\partial Eo1}{\partial netO1} = \frac{\partial Eo1}{\partial yo1} * \frac{\partial yo1}{\partial netO1} = 0.009$

$netH_2 = I_A*w12 + I_B*w22 + B = {}_{net} H_2 = 0* 0.24 + 1*-1.045 + 1 = -0.045$

$\frac{\partial netO1}{\partial aH2} = w12=0.24$

$$\frac{\partial Eo1}{\partial aH2} = 0.009 * 0.24 = 0.00216$$

$$\frac{\partial Eo2}{\partial aH2} = 1.18 * 0.24 = 0.2832$$

$$\frac{\partial E}{\partial aH2} = 0.28536$$

$$\frac{\partial aH2}{\partial netH2} = 0.49(1 - 0.49) = 0.2499$$

$\frac{\partial netH2}{\partial w12} = I_{A=0}$

$\frac{\partial E}{\partial w12} = 0$. Therefore, no change on weight

By applying the same methods w21 = 0.0993598 and new w21= 1.52 – 0.5(0.0993598) and w22= - 0.0023503095 and new w22 = -1.045 – 0.5(-0.0023503095)

Therefore,

W21 = 1.47

W22 = -1.043

Now let us check if new weights can improve the accuracy of an output by feeding inputs and weights net.

$H_1 = I_A*w11 + I_B*w21 + B, {}_{net} H_1 = 0* -0.72 + 1*1.47 + 1 = 2.47$

$netH_2 = I_A*w12 + I_B*w22 + B, {}_{net} H_2 = 0*0.24 + 1*-1.043 + 1 = -0.043$

$aH_1 = \frac{1}{1+e^{-netH1}} = \frac{1}{1+e^{-2.47}} = 0.92$

$aH_2 = \frac{1}{1+e^{-netH2}} = \frac{1}{1+e^{-(-0.043)}} = 0.48$

$netO_1 = aH_1*h11w + aH_2*h21w + B = 0.92*1.07 + 0.48*0.35 +1 = 2.15$

$netO_2 = aH1*h12w + aH_2*h22w + B = 0.92*-1.02+ 0.48*-0.32 +1 = -0.09$

$\frac{1}{1+e^{-netO1}} = \frac{1}{1+e^{-2.18}}$ = 0.898, 1.11= 0.900

$yo_2 = \frac{1}{1+e^{-netO2}} = \frac{1}{1+e^{0.18}}$ = 0.478

Previously actual output for O1 was 0.898 and after making adjustment on weight by back propagation in one iteration it become 0.900 and for O2 the output was 0.457 and it is now 0.478 and both current actual outputs are more closer to target output (which was 1 for both O1 and O2) than the previous one. By repeating the same back and forward propagation techniques many times the error will be minimized (as shown on Figure 4.6) and actual outputs will become equal or very close to target output. But while repeating the training, power and other resource constraints should be considered specially when implementing the network on embedded systems. We use Gaussian distribution techniques in assigning weight to minimize the risk.

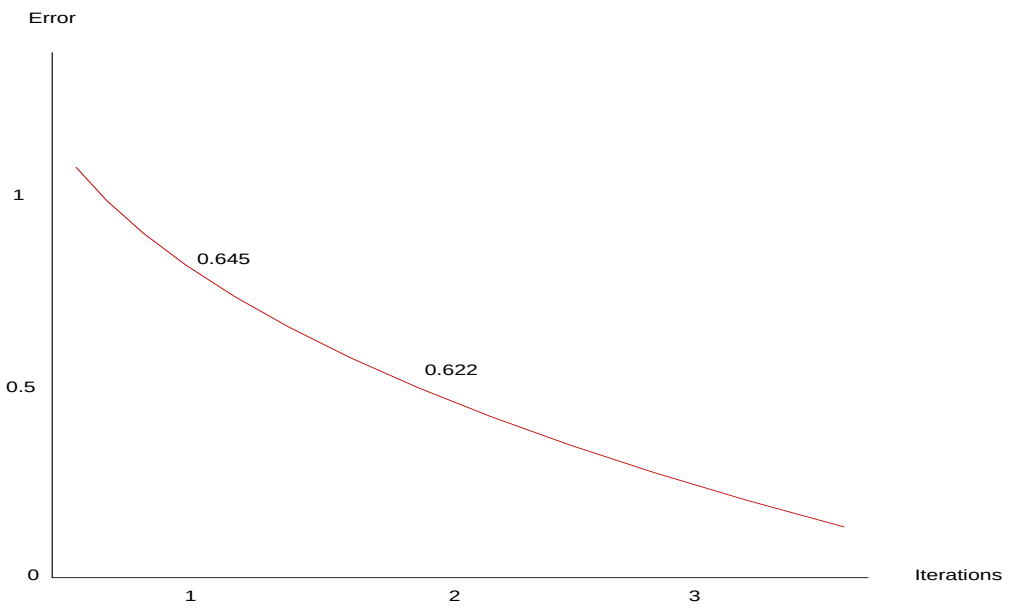

Figure 4.6: Error degradation with respect to iterations in adjusting weight

5. Experiment

5.1 Experimental Environment

We use C programming language to program our logic that is then embedded in the Arduino microcontroller. The C program on Arduino IDE and procedures which is simulated on Proteus is documented in Annexe A. The experiment is conducted on Windows 8, 2GB RAM, core i3 laptop.

5.2 Experimental Result

For input pattern 01 (as explained in section 4.1.2) the target output was 1 for both O1 and O2 and the actual output is 0.898 for O1 and 0.457 for O2 then after training it for the first iteration the output become 0.900 for O1 and 0.478 for O2 which is closer to the target output than the previous result therefore the error is decreased (from 0.153 to 0.141). By applying the same method, after 10 iterations actual output for O1 becomes 0.9983 for O1 and 0.9879 for O2. As number of iteration increase for training, the error will be closer to zero.

For input pattern 00 before training the network, the actual output for O1 was 0.1037 and -0.1006 for O2 and after training the network by adjusting its weight in 20 iterations actual outputs become 0.0020 and -0.0009 for O1 and O2 respectively.

Similarly, actual outputs for all patterns become closer to the target outputs as number of training cycle increase as shown in Figure 5.1.

A:	0	0	1	1
B:	0	1	0	1
Actual O1 before training:	0.1037	0.898	0.1003	0.0837
Actual O1 after training:	0.0020	0.9983	0.0033	0.0007
Target O1:	**0**	**1**	**0**	**0**
Actual O2 before training:	-0.1006	0.457	-0.0970	0.7496
Actual O2 after training:	-0.0009	0.9879	-0.0022	0.9998
Target O2:	**0**	**1**	**0**	**1**

Total Error before training:	0.0104	0.153	0.055	0.035
Total Error after training:	**0.0011**	**0.0138**	**0.0011**	**0.0005**
Number of Iteration:	20	10	15	30

Figure 5.1: Experimental result for context aware stove controller using embedded systems

When we compare the result with the result we obtained from the system before the training and after training, the result after training is much closer to the target output than the result before training for all input patterns and error is decreased as a result. After training the network, the result will be stored in a knowledgebase to react to inputs using the firing rule technique.

The result obtained from testing the deployed knowledgebase on arduino board for sensor reading are presented in Table 5.2. Two sensors was chosen for recognizing user existence (PIR sensor) and appliance hot level (LM35) .

Table 5.1: *System tests*

Test case		**Correct class**	**Predicted class**	**Success (%)**	**Response Time**
User existing	Normal hot level	A3	A3	100%	1 sec
User existing	Not Normal hot level	A2	A2	100%	7 sec
User not existing	Normal hot level	A3	A3	100%	5 sec
User not existing	Not Normal hot level	A1	A1	100%	3 sec

6. Conclusions and Future Work

In this thesis, lightweight neural network is used to solve context aware problems in embedded systems that can work for many applications. Information from sensors was recognized and it's passed to hidden layer with its effect to on the output. The training was done on high capacity computer before it is deployed on embedded device. Trained inputs and its respective output were stored in a knowledgebase. Then a knowledgebase along with the firing algorithm were deployed on embedded system for making smart decision based on contextualized inputs. Stove controller was used as scenario to evaluate the model. Three different sensors (temperature sensor, motion sensor and load sensor) were used for evaluating the training. The experimental result shows precise contextualization of inputs through tunning several parameters. Two heterogeneous sensors (temperature sensor and motion sensor) were used to validate the effectiveness of the model after knowledgebase and firing rule is deployed on the embedded device. Based on information from these sensors, input were contextualized and classified to correct actuation class in a real time. In general the NN proposed in this work can be applied in all embedded systems and other IoT applications that require context awareness. In addition, the work solves restriction of applying NN in resource limited embedded systems. However, neural network is limited to modelling complex problem solving numerically and it cannot model complex problems linguistically using natural language processing.

For the future work we plan to use fuzzy logic with neural network to take advantages of both fuzzy logic and neural networks; leading to context aware embedded systems that can: mimic the human decision-making process, handle vague information, learn by example (hence do not require the knowledge of a human expert) and process numeric, linguistic, or logical information.

References

[1] Ali Yavari," Contextualised Service Delivery in the Internet of Things," 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT).

[2] Glaisher, James Whitbread Lee "On a class of definite integrals". London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, July 1871.

[3]Leslie Pack KaelblingL, "Learning in Embedded Systems", The MIT Press, Cambridge University, London, England, 1993.

[4]Min-Jung Yoo "Real life applications of Embedded Systems ", 2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud).

[5] https://www.tutorialspoint.com/embedded_systems/es_overview.htm, last accessed on November 13, 2018.

[6] Susan Kurian and Michael J. Pont, "The maintenance and evolution of resource-constrained embedded systems created using design patterns", Embedded Systems Laboratory, University of Leicester, University Road, Leicester LE1 7RH, UK 2007.

[7]Dimitrios Serpanos and Tilman Wolf, "Architecture of embedded systems", Elsevier B.V, 2011.

[8]Sarah Vieweg," Rethinking Context: Leveraging Human and Machine Computation in Disaster Response", Computer, Apr. 2014.

[9] Sanja Fidler, "Neural networks", Shengyang Sun, 2017.

[10] https://www.tutorialspoint.com/artificial_neural_network, last accessed on 20 December, 2019

[11] Andrews, Larry C., " Special functions of mathematics for engineers", SPIE Press, 1998.

[12] Alex Smola and S.V.N. Vishwanathan , "Introduction to Machine Learning", Cambridge University Press, Cambridge, United Kingdom, 2010.

[13] S. Bashyal, G. K. Venayagamoorthy, and B. Paudel, "Embedded neural network for fire classification using an array of gas sensors," in Sensors Applications Symposium, 2008.SAS 2008. IEEE, 2008, pp. 146-148.

[14] https://www.mathsisfun.com/data/standard-normal-distribution.html, last accessed on May 10, 2019.

[15] Vivian GenaroMotti, "Machine Learning in the Support of Context-Aware Adaptation", Université catholique de Louvain, 2017.

[16] Christos Stergiou and DimitriosSiganos ,"Neural Networks", Computer Science & Engineering: An International Journal (CSEIJ), Vol. 4, No. 1, February 2014

[17] Horvitz , "The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users", Proceedings of the Fourteenth Conference on Uncertainty in AI: Microsoft Research Redmond, WA 98052-6399.

[18] J Sathish Kumar, "Forward and Back propagation" Computer Engineering Department, Surat, India,2016.

[19] Jennings, A. and Higuchi, "A User Model Neural Network for a Personal News Service", User Modeling and User-Adapted Interaction 3, S. 1–25, 2017.

[20] Mitrovic, N. Royo, J. A. and Mena, "Adaptive Graphical User Interfaces: An Approach Based on Mobile Agents", Networks, pp. 1-36, 2009.

[21] Kwang-EunKo, "Development of Context Aware System based on Bayesian Network driven Context Reasoning Method and Ontology Context Modeling", School of Electrical and Electronics Engineering, Chung-Ang University, Seoul 156-756, Korea.

[22] Eric Horvitz, "Bayesian User Modeling for Inferring the Goals and Needs of Software Users", Microsoft Research, Redmond, 2016.

[23] Hai-taoZheng, Bo-Yeong Kang, Hong-Gee Kim, "An Ontology-based Bayesian Network Approach for Representing Uncertainty in Clinical Practice Guidelines", 6th International Semantic Web Conference 2007 Workshop on Uncertainty Reasoning for the Semantic Web.

[24]Smith, A. S. G. and Blandfor, "An Application of Machine Learning Algorithms for an Adaptive Web-based Information System", International Journal of Artificial Intelligence in Education, pp. 1-22, 2002.

[25] Carmen De Maio, "A Context-aware Fuzzy Linguistic Consensus Model supporting Innovation Processes", 2016 IEEE International Conference on Fuzzy Systems (FUZZ)

[26] Toshihiro Hattori, "Challenges for Low-power Embedded SOC"', Josuihon-cho, Kodaira Tokyo Japan, 2007.

[27] Sonal Linda, "A Decision Tree Based Context-Aware Recommender System", Springer Nature Switzerland AG 2018 U. S. Tiwary.

[28] Ahmad Faridi, "HMM as an Inference Technique for Context Awareness", Department of Computer Science, IIUM, Kuala Lumpur and 50728, Malaysia.

[29] Tsungnanlin, "A neural-network-based context-aware handoff algorithm for multimedia computing, acm transactions on multimedia computing, communications and applications", vol. 4, no. 3, article 17, August 2018.

[30] Gu T., Pung H.K. and Zhang D.Q., "A Bayesian approach for dealing with uncertain contexts", In the Proceeding of the Second International Conference on Pervasive Computing (Pervasive 2014), Vienna, Austria, April 2014.

[31] Daniel L., "Ontology-Based Modeling of Context-Aware Systems", Volkswagen AG, Group Research, 38440 Wolfsburg, Germany, 2014.

[32] Ye Ning, "Fuzzy Logic Based Middleware Approach for Context Processing", International Journal of Digital Content Technology and its Applications Volume 3, Number 3, September 2017.

[33] Hongyuan W., "Rule-based context-aware adaptation: a goal-oriented approach", International Journal of Pervasive Computing and Communications Vol. 8 No. 3, 2012

[34] https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks, last accessed on Aril 6, 2019.

[35] WouterGevaert and GeorgiTsenov, “Neural networks used for speech recognition”, JOURNAL OF AUTOMATIC CONTROL,UNIVERSITY OF BELGRADE,VOL.20:1-7,2010.

[36] Yoav Goldberg, Neural Network Methods for Natural Language Processing, Morgan and Claypool publisher, April 2017.

[37] https://www.codementor.io/james_aka_yale/a-gentle-introduction-to-neural-networks-for-machine-learning-hkijvz7lp , last accessed on June 10, 2019.

[38] Anil Suryavanshi and ShivnandanMandre, “Hidden Markov Model based Framework for User’s Uniqueness in Pervasive Computing”, International Journal of Computer Applications (0975 – 8887) Volume 156 – No 14, December 2016.

[39] Thomas A. Henzinger and Joseph Sifakis, “The Embedded Systems Design Challenges”, Springer-Verlag Berlin Heidelberg 2006.

Annexes

Annex A – c program that is simulated on proteus

```
intinputA = A0;

intinputB = 13;

intinputW = A1;

//intsigned_inputA ;

//intsigned_inputB ;

floatinputAVal;

floatinputWVal;

floatinputAWeight[2];  //w11, w12

floatinputBWeight[2];  //w21, w22

floatinputW_Weight[2];// w31, w32


float layer1_1Val;

float layer1_2Val;


float layer1_1Weight[2];  //h11w, h12w

float layer1_2Weight[2];  // h21w, h22w


float output1;

float output2;


float learningRate = 0.5;

int randomWeight;

int randomWeightVal;  //by guassian distribution
```

```
int lastChange;
float error;
float lastError;
float currentInputWVal;
float lastWeight;
int TREAT;
int trainCycle;

void setup()
{
Serial.begin(9600);
pinMode(A0, INPUT);
pinMode(A1, INPUT);
pinMode(13, INPUT);
generateSignedInput();
generateCurrentInputWeights();
generateInputWeights();
generateLayer1Weights();
}
void loop()
{
inputAVal = anaglogRead(A0);
inputWVal = anaglogRead(inputW);
inputBVal = digitalRead(13);
```

```
inputMapAndConvert();
layer1Val();
output1 = layer1_1Val * layer1_1Weight[0] + layer1_2Val * layer1_2Weight[0] ;
 output2 = layer1_1Val * layer1_1Weight[1] + layer1_2Val * layer1_2Weight[1];
 output1 = 1 / (1 + exp(-1 * output1));
 output2 = 1 / (1 + exp(-1 * output2));
Serial.print("TREAT  ");
Serial.print(TREAT);
Serial.print("     ");
Serial.print("Training Cycle  ");
Serial.println(trainCycle);
}
currentInputWVal = inputWVal;
voidgenerateSignedInput()
{
inputAVal = -1*inputAval;
inputBVal = 1*inputBval;
inputWVal = 1*inputAval;
}
generateCurrentInputWeights(){
lastWeight = inputWVal-currentInputWVal;
}
voidgenerateInputWeights()
{
```

```
//generate input A weights
for (int x = 0; x <= 1; x++)
 {
inputAWeight[x] = random(1, 101);
 }
 //generate input B weights
for (int x = 0; x <= 1; x++)
 {
inputBWeight[x] = random(1, 101);  //101 because random number generator
//subtracts 1 from max.
 }
//generate input W  weights
for (int x = 0; x <= 1; x++)
 {
inputWWeight[x] = random(1, 101);
 }

 //-----convert input weights to float values---
for (int x = 0; x <= 1; x++)
 {
inputAWeight[x] = inputAWeight[x] / 100;
 }
for (int x = 0; x <= 1; x++)
 {
inputBWeight[x] = inputBWeight[x] / 100;
```

```
  }
for (int x = 0; x <= 1; x++)
  {
inputW_Weight[x] = inputW_Weight[x] / 100;
  }


}
void generateLayer1Weights()
{
//generate the first hidden layer (H1) weights
for (int x = 0; x <= 1; x++)
  {
layer1_1Weight[x] = random(1, 101); //101 because random number generator subtracts 1 //from max.
  }
  // generate the second hidden layer (H2) weights
for (int x = 0; x <= 1; x++)
  {
layer1_2Weight[x] = random(1, 101);
  }
  //convert layer values to decimal
for (int x = 0; x <= 1; x++)
  {
    layer1_1Weight[x] =  layer1_1Weight[x] / 100;
  }
```

```
for (int x = 0; x <= 1; x++)
 {
   layer1_2Weight[x] =  layer1_2Weight[x] / 100;
 }
}
void training()
{
trainCycle++;
error = output1 - output2;
if (error <lastError)
 {
lastError = error;
randomWeight = random(1, 9);
if (randomWeight == 1)
   {
randomWeightVal = random(0, 3);
inputAWeight[randomWeightVal] = inputAWeight[randomWeightVal] + learningRate;
   }
if (randomWeight == 2)
   {
randomWeightVal = random(0, 3);
inputBWeight[randomWeightVal] = inputBWeight[randomWeightVal] + learningRate;
   }
```

```
if (randomWeight == 3)
  {
randomWeightVal = random(0, 3);
inputW-Weight[randomWeightVal] = inputW-Weight[randomWeightVal] + learningRate;
  }

if (randomWeight == 4)
  {
randomWeightVal = random(0, 2);
   layer1_4Weight[randomWeightVal] = layer1_4Weight[randomWeightVal] + learningRate;
  }
if (randomWeight == 5)
  {
randomWeightVal = random(0, 2);
   layer1_4Weight[randomWeightVal] = layer1_4Weight[randomWeightVal] + learningRate;
  }
if (randomWeight == 6)
  {
randomWeightVal = random(0, 2);
   layer1_4Weight[randomWeightVal] = layer1_4Weight[randomWeightVal] + learningRate;
  }
```

```
if (randomWeight == 7)
  {
randomWeightVal = random(0, 2);
   layer1_4Weight[randomWeightVal] = layer1_4Weight[randomWeightVal] + learningRate;
  }
 }
if (error >lastError)
 {
if (randomWeight == 1)
  {
randomWeightVal = random(0, 3);
inputAWeight[randomWeightVal] = inputAWeight[randomWeightVal] - (learningRate * 2);
  }
if (randomWeight == 2)
  {
randomWeightVal = random(0, 3);
inputBWeight[randomWeightVal] = inputBWeight[randomWeightVal] - (learningRate * 2);
  }
if (randomWeight == 3)
  {
randomWeightVal = random(0, 2);
   layer1_4Weight[randomWeightVal] = layer1_4Weight[randomWeightVal] -
```

```
(learningRate * 2);
   }
if (randomWeight == 4)
   {
randomWeightVal = random(0, 2);
    layer1_4Weight[randomWeightVal] = layer1_4Weight[randomWeightVal] -
(learningRate * 2);
   }
if (randomWeight == 5){
randomWeightVal = random(0, 2);
    layer1_4Weight[randomWeightVal] = layer1_4Weight[randomWeightVal] -
(learningRate * 2);
   }
if (randomWeight == 6){
randomWeightVal = random(0, 2);
    layer1_4Weight[randomWeightVal] = layer1_4Weight[randomWeightVal] -
(learningRate * 2);
   }
 }
}
voidinputMapAndConvert(){
//make LM35  values in  %C
input1Val = (input1Val/1024.0)*5000;
//make the sensor to pass 1 if the temperature is greater than 150 %C otherwise 0
if(input1Val>=150)
```

```
input1Val = 1;

else

input1Val = 0;

//pass one from motion sensor if user activity is detected otherwise 0;

if(input2Val == HIGH)

input2Val = 1;

else

input2Val = 0;

//make weight sensor values between 0 and 1000.

inputWVal = map(input1Val, 0, 1023, 0, 1000);

//check there is a change between previous and current, pass 1 if there is a change otherwise 0.

if(lastWeight>1)

inputWVal =1;

else

inputWVal =0;

}

void layer1Val(){

//determine layer 1 values based upon inputs and weights.

 layer1_1Val = SignedInputAVal * inputAWeight[0] + SignedInputBVal * inputBWeight[0] + SignedInputWVal * inputW_Weight[0]

 layer1_2Val = SignedInputAVal * inputAWeight[1] + SignedInputBVal * inputBWeight[1] + SignedInputWVal * inputW_Weight[1]

 layer1_1Val = 1 / (1 + exp(-1 * layer1_1Val));

 layer1_2Val = 1 / (1 + exp(-1 * layer1_2Val));
```

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other university, and that all source of materials used for the thesis have been duly acknowledged.

Declared by:

Name: _____

Signature: _____

Date: _____

Confirmed by advisor:

Name: _____

Signature: _____

Date: _____