Lightweight Neural Networks for Context Aware Embedded System

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC BY 4.0

SUBMISSION DATE / POSTED DATE

05-10-2021 / 16-10-2021

CITATION

Dera, Abdi (2021): Lightweight Neural Networks for Context Aware Embedded System. TechRxiv. Preprint.
<https://doi.org/10.36227/techrxiv.16743901.v1>

DOI

[10.36227/techrxiv.16743901.v1](https://doi.org/10.36227/techrxiv.16743901.v1)

Lightweight Neural Networks for Context Aware Embedded System

Abdi Mosisa, Abebe Diro

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 paper 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. 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.

Index Terms—Embedded Systems, Context awareness, Neural Networks, Internet of Things

1 INTRODUCTION

EEmbedded 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-dimensionally ranging from industrial machines to home appliances. It can also play significant role in building smart such as smart cities 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 deals with 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 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]. 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.

2 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 that effectively considers the entire range of data that is being collected in smart cities and use such data to provide information that best suits the context of each user in the Smart City was proposed. 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.

Ahmad 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 a 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 $\rightarrow$ Action

"Adaptation" rules take the form:

Contextual condition $\rightarrow$ Software system action

Rules for "Context" state change would be:

Condition and action $\rightarrow$ 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 its 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 paper focuses on filling the gap in using heterogeneous input sensors and deploying neural networks on resource limited embedded devices.

2 PROPOSED SYSTEM

2.1 Architecture

In the domain of context awareness, a technology cannot be described as context aware without at least exhibiting the ability to detect a 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 this case, NN is utilized at the heart of embedded systems (as shown in Figure 4.1) as our inference techniques of choice. The scenario of stove controller will be used as one

of the widely used home appliances.

In the design of neural network for context aware embedded system, there are input elements (sensors) that have direct relationship with the output (actuation) and some of them have reverse or indirect relationship. In addition to the 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

As shown in fig. 4.1, input from different sensors will be passed to the hidden layer of the neural network. To calculate the input value, parameters namely 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 design, weight of neuron in the input layer and hidden layer is assigned by using Gaussian 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 computations 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.

The NN has been designed in such a way the actuation is made without training the network, which reduces computational expense (both memory and time). The firing algorithm classifies all inputs (including the untrained inputs) to actuation class is used to make the embedded system to be flexible in dynamic environments. These design choices enabled to build a learning system which can fit into the requirements of resource-constrained embedded devices in the context-aware environment.

Considering smart kitchen as a scenario, increase in weight of input from temperature sensor of a stove has direct relationship with actuation to help the decision of switching 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 the load of the stove has also directly proportional to the actuation and it is calculated as $\Delta W_t = w_{t_i} - w_{t_c}$ where, ΔW_t is change in load, w_{t_i} is initial load and w_{t_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

As shown in fig. 4.2, LM35, PIR and HX711 are temperature, motion and load sensors respectively, while S_1 , S_2 , and S_3 are sensor states (-1, 0 or 1). H_1 and H_2 are hidden nodes taking respective weights of w_{11} , w_{12} , w_{21} , w_{22} , w_{31} , and w_{32} whereas $h_{11}w$, $h_{12}w$, $h_{21}w$, and $h_{22}w$ are weights from hidden layers for the output nodes O_1 and O_2 . The scenario has B as a bias input, $f(x)$ as activation function, variables A_1 , A_2 and A_3 as actuation classes i.e. A_1 switching off, A_2 alert, A_3 normal, and KB as a knowledgebase which is used to store output for trained inputs. F is firing rule algorithm which classifies the contextual information into actuation class based on the knowledgebase. The broken lines represent novel design concept employed in this paper, and others are taken from neural network design [37].

2.2 Firing Rule for the proposed design

The adopted Firing rule algorithm is different from other technique such as Hamming distance technique in firing options. In the applied technique, in addition to firing and not to fire, how far the input is from firing and not firing is known and based on the distance the output can be classified into one of the intermediate classes. It takes 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 the 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 A_1 when the input (X_1 , X_2 and X_3) is 111 and A_3 when the input is 000. Then, by applying the firing algorithm in every column the following truth table is obtained for three actuation classes A_1 , A_2 , A_3 ;

Table 4.1: Sample firing rule technique for the proposed solution

X_1	X_2	X_3	Class
0	0	0	A_3
0	0	1	A_2
0	1	0	A_2
0	1	1	A_2
1	0	0	A_2
1	0	1	A_2
1	1	0	A_2
1	1	1	A_1

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 from the knowledge. However, the algorithm has taken designed to enable NN 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 to 111 patterns. To determine firing class next to either A_1 or A_3 , the formula $N-m$ has been used. For the three classes in

the table, $N=2$, and the two similar patterns in classes gives $m=2$. Hence, intermediate firing class will be the first (3-2) class next to A_1 which means class A_2 will be fired. The same is true for other patterns.

2.3 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]

In forward propagation, a set of weights has been applied to the input data and calculate an output. For the initial forward propagation, the set of weights has been assigned by Gaussian distribution. The sensors values have been used as input for the hidden layers, and outputs from hidden layers have been input for output node/s. In addition to input values, the effect of the sensor should also be known for sensors. To do these, the following procedures have been carried out as described in Algorithm 4.1.

First, input from each sensor is multiplied by a state as $I_i = X_i \cdot S_i$, where $X_i = x_1, x_2, x_3, \dots, x_n$ are inputs and $S_i = s_1, s_2, s_3, \dots, s_n$ are states and I_i are signed inputs. Then, in our scenario, the signed inputs are computed as $I_1 = LM35 \cdot s_1$, $I_2 = PIR \cdot s_2$ and $I_3 = HX711 \cdot s_3$, and then, all signed inputs are multiplied by a weight as

$$I_1 \rightarrow I_1 \cdot w_{11}, I_1 \cdot w_{12}$$

$$I_2 \rightarrow I_2 \cdot w_{21}, I_2 \cdot w_{22}$$

$$I_3 \rightarrow I_3 \cdot w_{31}, I_3 \cdot w_{32}$$

Thirdly, all weighted signed inputs for each hidden nodes are added together with a bias as $[H_i] = I_i \cdot [W] + B$, where $H_i = H_1, H_2, H_3, \dots$, $W = w_{11}, w_{12}, w_{21}, w_{22}, \dots$

$$[H_1, H_2] = [I_1 \quad I_2 \quad I_3] \cdot \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} + B$$

$$H_1 = (I_1 \cdot w_{11} + I_2 \cdot w_{21} + I_3 \cdot w_{31}) + B$$

$$H_2 = (I_1 \cdot w_{12} + I_2 \cdot w_{22} + I_3 \cdot w_{32}) + B$$

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

$$y_1 = f(H_1)$$

$$y_2 = f(H_2)$$

where, y_1 and y_2 are outputs from hidden layers.

y_1 and y_2 are outputs from hidden layers (H_1 and H_2 respectively) and next it will be input for output node.

The following procedures have been followed to compute output for the output node.

First, each value from hidden layers are multiplied by a weight as

$$y_1 \rightarrow y_1 \cdot h_{11}w, y_1 \cdot h_{21}w$$

$$y_2 \rightarrow y_2 \cdot h_{12}w, y_1 \cdot h_{22}w$$

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

$$O_1 = (y_1 \cdot h_{11}w + y_2 \cdot h_{21}w) + B \text{ and } O_2 = (y_1 \cdot h_{12}w + y_2 \cdot h_{22}w) + B$$

Finally, the sum is passed through an activation function as

$$y_{O1} = f(y_1 \cdot h_{11}w + y_2 \cdot h_{21}w + B)$$

$$y_{O2} = f(y_1 \cdot h_{12}w + y_2 \cdot h_{22}w + B)$$

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]:

$$\text{Sigmoid}(z) = \frac{1}{1 + e^{-z}}$$

where, z is an output of hidden nodes and output nodes. h_1 's output: $z = I_1 \cdot w_{11} + I_2 \cdot w_{21} + B = H_1$, h_2 's output: $z = (I_1 \cdot w_{12} + I_2 \cdot w_{22}) + B$, O_1 's output: $y_1 \cdot h_{11}w + y_2 \cdot h_{21}w + B$ and O_2 's output: $y_1 \cdot h_{12}w + y_2 \cdot h_{22}w + B$.

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.

The 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 gaussian
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

The forward propagation algorithm has been uniquely designed while the backward propagation has been adapted from [18]. In back propagation, the margin of error of the output is measured, and the weights are adjusted accordingly to decrease the error. Error is initially calculated by computing the difference between actual outputs (forward propagation) and the target output (true output), as described in Algorithm 4.2. Error in each output nodes is calculated by mean square error (MSE) [11] as $EO_i = 1/2(\text{target output} - \text{actual output})^2$. Total error can be computed as $E_{\text{total}} = \sum 1/2(\text{target output} - \text{actual output})^2$. The 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 [2] as $\partial E / (\partial w^{k_{ij}})$, 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 $\partial E / (\partial w^{k_{ij}}) = \partial E / (\partial y_{oi}^k) * (\partial y_{oi}^k) / (\partial w^{k_{ij}})$, where y_{oi} is an actual output in k layer. This can be further decomposed as $\partial E / (\partial w^{k_{ij}}) = (\partial \sum 1/2 ((\text{target output} - \text{actual output}))^2 / (\partial y_{oi}^k) * (\partial y_{oi}^k) / (\partial w^{k_{ij}})$. Computation of the error terms will proceed backwards from the output layer down to the input layer.

In output layer, how much a change in w_{ij} affects the total error $\partial E / (\partial w^{k_{ij}})$ should be known using a chain rule

$\partial E / (\partial w^{k_{ij}}) = \partial E / (\partial y_{oi}^k) * (\partial y_{oi}^k) / \partial \text{net}_{oi} * \partial \text{net}_{oi} / (\partial w^{k_{ij}})$. Next, how much does the output of oi change with respect to its total net input should be known. The partial derivative of the sigmoid function is the output multiplied by 1 minus the output [2] $y_{oi} = 1 / (1 + e^{-(\text{net}_{oi})})$, where o_i is net input which is weighted sum

$(\partial y_{oi}) / \partial o_i = y_{oi}(1 - y_{oi})$. Finally, how much does the total net input of oi change with respect to w_{ij} is computed as-

$$(\partial \text{net}_{oi}) / \partial w_{ij}.$$

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

Alternatively, delta rule can be used to calculate $(\partial E) / \partial \text{net}_{oi}$ as $\delta o_i = (\partial E) / \partial y_{oi} * (\partial y_{oi}) / \partial \text{net}_{oi} = (\partial E) / \partial \text{net}_{oi}$, where, δ is delta. Hence, $\delta o_i = -(\text{target}_{oi} - \text{actual}_{oi}) * \text{actual}_{oi}(1 - \text{actual}_{oi})$. Therefore, $(\partial E) / (\partial w^{k_{ij}}) = \delta o_i \cdot \text{actual}_{hi} - 1$, where k is network layer. To decrease the error, this value is subtracted from the current weight as:

$$\text{New weight} = \text{old weight} - (\partial E) / (\partial w^{k_{ij}})$$

$$\text{New } w_{ij} = w_{ij} - (\partial E) / (\partial w^{k_{ij}})$$

Optionally the delta is multiplied by some learning rate, which we'll set to 0.5 as $w_{ij} = w_{ij} - \Theta \cdot (\partial E) / (\partial w^{k_{ij}})$. Learning rate is mainly used for controlling how much the weights of the network is adjusted with respect to the loss gradient. The actual updates in the neural network are performed after 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) are obtained.

In hidden Layer, the backwards pass will continue by calculating new values for weights leading into the next hidden layer neurons. This can be done as $(\partial E) / (\partial w^{k_{ij}}) = (\partial E) / \partial a_{hi} * (\partial a_{hi}) / \partial \text{net}_{hi} * (\partial \text{net}_{hi}) / (\partial w^{k_{ij}})$, where, a_{hi} is actual output in for hidden layer neuron i , net_{hi} is net input to hidden neuron which is weighted sum leading into the neuron. A similar process is used as the output layer, differing only in that the output of each hidden layer neuron contributes to the output of multiple output neurons. For instance, a_{hi} affects all y_{oi} . Therefore, the $(\partial$

$E/\partial aH_i$ needs to take into consideration its effect on the output neurons $(\partial E)/\partial aH_i = (\sum \partial E_{y_{oi}})/\partial aH_i$, where $E_{y_{oi}}$ is error in each output neurons $(\partial E_{y_{oi}})/\partial y_{oi} = (\partial E_{y_{oi}})/\partial net_{oi} * (\partial net_{oi})/\partial y_{oi}$. $(\partial E_{y_{oi}})/\partial net_{oi}$ as can be calculated as $(\partial E_{y_{oi}})/\partial net_{oi} = (\partial E_{y_{oi}})/\partial y_{oi} * (\partial y_{oi})/\partial net_{oi}$ since $(\partial net_{oi})/\partial y_{oi}$ is known. Thus, $(\partial E)/\partial aH_i = \sum (\partial E_{y_{oi}})/\partial aH_i$.

Having $(\partial E)/\partial aH_i$, it needs to figure out $(\partial aH_i)/\partial netH_i$ and $(\partial netH_i)/\partial w_{ij}$ for each weight $aH_i = 1/(1+e^{-(netH_i)})$ $(\partial aH_i)/\partial netH_i = aH_i(1-aH_i)$. The partial derivative of the total net input to H_i with respect to w_{ij} can be calculated the same way as it is calculated for the output neuron $(\partial netH_i)/\partial w_{ij} = \text{weighted sum leading into } H_i$. Finally total error with respect to weight is computed as $(\partial E)/\partial w_{ij} = (\partial E)/\partial aH_i * (\partial aH_i)/\partial netH_i * (\partial netH_i)/\partial w_{ij}$. The weight $e w_{ij}$:

$w_{ij} = w_{ij} - \theta * (\partial E)/\partial w_{ij}$ can now be updated. 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
  // Propagate 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
    while ((maximum number of iterations < than specified)
    AND
    (Error Function is > than specified))

```

Neural networks repeat both forward and back

propagation until the weights are calibrated to accurately predict an output.

2.4 Evaluation

During training phase, the network inputs are first fed by forward propagation. In addition to the input values, states of inputs are be fed before assigning weights to inputs. The assignment of weights is taken place by using Gaussian distribution rather than random assignment as shown in Figure 4.5. This can minimize iterations for updating weight (therefore optimize resource constraints) than random assignment of weights. For instance, if 2, 5, 9, and 1 are assigned for w_{11} , w_{12} , w_{21} , and w_{22} respectively, then by using Gaussian distribution the value for w_{11} will be $(2-\text{mean})/\text{stdv}$, where stdv is standard deviation. This means having $\text{stdv}=3.11$ and $\text{mean}=4.25$, thus the value of w_{11} will be $w_{11} = -2.25/3.11 = -0.72$. Similarly, $w_{12} = 0.24$, $w_{21} = 1.52$ and $w_{22} = -1.045$. Let us assign state -1 for S_1 and state 1 for S_2 . The value for bias in all neuron is 1. If weight 4, 1, 3, and 2 are assigned for $h_{11}w$, $h_{12}w$, $h_{21}w$, and $h_{22}w$, respectively, by applying similar method in hidden layer, $h_{11}w = 1.08$, $h_{12}w = -1.08$, $h_{21}w = 0.36$, $h_{22}w = -0.36$.

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

Now, input is fed to two neurons (in our case LM35, PIR) in input layer (we skip HX711 for simplicity) then give target outputs for all input patterns in 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 instance, if 01 pattern is taken for inputs, the target output will be 11. To get the actual output, 5 procedures are carried out. First, 0 is multiplied by -1 to become 0 and 1 is multiplied by 1 to become 1. This means, the input now becomes 01, however, it's not always the same for all input patterns as a sign can sometimes be changed based on the state. Next, the signed input from neurons is multiplied by respective weights to get net input for hidden neuron $H_1 = I_A * w_{11} + I_B * w_{21} + B_{netH_1} = 0 * -0.72 + 1 * 1.52 + 1 = 2.52$, and $netH_2 = I_A * w_{12} + I_B * w_{22} + B_{netH_2} = 0 * 0.24 + 1 * -1.045 + 1 = -0.045$. Thirdly, actual output for hidden neuron is then calculated by sigmoid activation function as

$$aH_1 = \frac{1}{1+e^{-netH_1}} = \frac{1}{1+e^{-2.52}} = 0.93$$

$$aH_2 = \frac{1}{1+e^{-netH_2}} = \frac{1}{1+e^{-(-0.045)}} = 0.49$$

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

Fourthly, net input for output neuron is calculated as $netO_1 = aH_1 * h_{11w} + aH_2 * h_{21w} + B_{netO_1} = 0.93 * 1.08 + 0.49 * 0.36 + 1 = 2.18$

$netO_2 = aH_1 * h_{12w} + aH_2 * h_{22w} + B_{netO_2} = 0.93 * -1.08 + 0.49 * -0.36 + 1 = -0.18$

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

$$yO_1 = \frac{1}{1+e^{-netO_1}} = \frac{1}{1+e^{-2.18}} = 0.898$$

$$yO_2 = \frac{1}{1+e^{-netO_2}} = \frac{1}{1+e^{-0.18}} = 0.457$$

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

In calculating the total error, the error for each output neuron can be calculated as $E_{total} = \sum_2^1 (t_o - y_o)^2$ where, t_o is target output and y_o is actual output.

$$E_{O1} = \frac{1}{2} (t_{O1} - y_{O1})^2 = \frac{1}{2} (1 - 0.898)^2 = 0.0052$$

$$E_{O2} = \frac{1}{2} (t_{O2} - y_{O2})^2 = \frac{1}{2} (1 - 0.457)^2 = 0.1474$$

Thus, $E_{total} = E_{O1} + E_{O2} = 0.1526$

In back propagation, the goal 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).

In calculating weight in output layer, such as H_{11w} , the extent of change in H_{11w} should be known for the effects it can bring about in the total error, $\frac{\partial E}{\partial H_{11w}}$. It can be calculated by delta rule as

$$\frac{\partial E}{\partial H_{11w}} = -(target_{O1} - actual_{O1}) * actual_{O1} (1 - actual_{O1}) * aH_1$$

$$d_{O1} = \frac{\partial E}{\partial actual_{O1}} * \frac{\partial actual_{O1}}{\partial netO1} = \frac{\partial E}{\partial netO1}$$

$$d_{O1} = -(target_{O1} - actual_{O1}) * actual_{O1} (1 - actual_{O1})$$

Hence, $\frac{\partial E}{\partial H_{11w}} = -d_{O1} * aH_1 = -(-(1 - 0.898) * 0.898 (1 - 0.898)) * 0.93 = 0.0087$. To decrease the error, this value

should be subtracted from weight as $H_{11w} = H_{11w} - \theta \frac{\partial E}{\partial H_{11w}}$ where θ is learning rate and its value is 0.5.

$H_{11w} = 1.08 - 0.5 * 0.0087 = 1.07$. Therefore new weight for H_{11w} is 1.07

Similarly,

$$\frac{\partial E}{\partial H_{21w}} = -d_{O1} * aH_2$$

$$\frac{\partial E}{\partial H_{21w}} = -(-(1 - 0.898) * 0.898 (1 - 0.898)) * 0.49 = 0.0046$$

$$H_{21w} = H_{21w} - \theta \frac{\partial E}{\partial H_{21w}} = 0.36 - 0.0023 = 0.35$$

$$\frac{\partial E}{\partial H_{12w}} = -d_{O2} * aH_1$$

$$d_{O2} = -(target_{O2} - actual_{O2}) * actual_{O2} (1 - actual_{O2}) * aH_1 = -(1 - 0.457) * 0.457 (1 - 0.457) * 0.93 = -0.125$$

$$H_{12w} = H_{12w} - \theta \frac{\partial E}{\partial H_{12w}} = -1.08 - 0.0625 = -1.02$$

$$\frac{\partial E}{\partial H_{22w}} = -d_{O2} * aH_2$$

$$d_{O2} = -(target_{O2} - actual_{O2}) * actual_{O2} (1 - actual_{O2}) * aH_2 = -(1 - 0.457) * 0.457 (1 - 0.457) * 0.49 = -0.066$$

$$H_{22w} = H_{22w} - \theta \frac{\partial E}{\partial H_{22w}} = -0.36 - 0.033 = -0.32$$

Weight in hidden layer

Next, we calculate new values for w_{11} , w_{12} , w_{21} and w_{22} . When we start with w_{11} :

$$\frac{\partial E}{\partial w_{11}} = \frac{\partial E}{\partial aH_1} * \frac{\partial aH_1}{\partial netH_1} * \frac{\partial netH_1}{\partial w_{11}}$$

$$\frac{\partial E}{\partial aH_1} = \frac{\partial E_{O1}}{\partial aH_1} + \frac{\partial E_{O2}}{\partial aH_1}$$

$$\frac{\partial E_{O1}}{\partial aH_1} = \frac{\partial E_{O1}}{\partial netO1} * \frac{\partial netO1}{\partial aH_1}$$

$$\frac{\partial E_{O1}}{\partial netO1} = \frac{\partial E_{O1}}{\partial y_{O1}} * \frac{\partial y_{O1}}{\partial netO1}$$

$$\frac{\partial E_{O1}}{\partial y_{O1}} = -0.102, \frac{\partial y_{O1}}{\partial netO1} = 0.091$$

$$\frac{\partial E_{O1}}{\partial netO1} = 0.009$$

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

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

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

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

$$\begin{aligned}\frac{\partial Eo2}{\partial aH1} &= 0.134745993 * -1.1808 = \\ &0.15910806853\end{aligned}$$

Therefore,

$$\begin{aligned}\frac{\partial E}{\partial aH1} &= 0.0106272 + 0.15910806853 \\ &= 0.1697352685\end{aligned}$$

$$\begin{aligned}\frac{\partial aH1}{\partial netH1} &= aH1(1 - aH1) = 0.93(1 - 0.93) = \\ &0.651\end{aligned}$$

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

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

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

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

For weight w12:

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

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

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

$$\frac{\partial Eo1}{\partial netO1} = \frac{\partial Eo1}{\partial yO1} * \frac{\partial yO1}{\partial netO1} = 0.009$$

$$netH2 = I_A * w12 + I_B * w22 + B =_{net} H2 = 0 * 0.24 + 1 * -1.045 + 1 = -0.045$$

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

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

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

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

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

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

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

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,

$$W_{21} = 1.47$$

$$W_{22} = -1.043$$

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

$$H1 = I_A * w11 + I_B * w21 + B, \quad_{net} H1 = 0 * -0.72 + 1 * 1.47 + 1 = 2.47$$

$$netH2 = I_A * w12 + I_B * w22 + B, \quad_{net} H2 = 0 * 0.24 + 1 * -1.043 + 1 = -0.043$$

$$aH1 = \frac{1}{1 + e^{-netH1}} = \frac{1}{1 + e^{-2.47}} = 0.92$$

$$aH2 = \frac{1}{1 + e^{-netH2}} = \frac{1}{1 + e^{-(-0.043)}} = 0.48$$

$$netO1 = aH1 * h11w + aH2 * h21w + B = 0.92 * 1.07 + 0.48 * 0.35 + 1 = 2.15$$

$$netO2 = aH1 * h12w + aH2 * h22w + B = 0.92 * -1.02 + 0.48 * -0.32 + 1 = -0.09$$

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

$$y_{O2} = \frac{1}{1+e^{-netO2}} = \frac{1}{1+e^{0.18}} = 0.478$$

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

3 EXPERIMENTATION

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.

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 O ₁ before training:	0.1037	0.898	0.100	0.083
Actual O ₁ after training:	0.002	0.998	0.003	0.000
Target O ₁ :	0	1	0	0
Actual O ₂ before training:	-0.1006	0.457	-0.0970	0.749
Actual O ₂ after training:	-0.0009	0.987	-0.0022	0.999
Target O ₂ :	0	1	0	1
Total Error before training:	0.0104	0.153	0.055	0.035
Total Error after training:	0.0011	0.013	0.001	0.000
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
Existing	Use Nor	A ₃	A ₃	100 %	1 sec
	mal hot level				
Existing	Use Not	A ₂	A ₂	100 %	7 sec
	Normal hot level				
Existing	Use Nor	A ₃	A ₃	100 %	5 sec
	mal hot level				

Use	Not				
r not	Norma	A_1	A_1	100	3 sec
existin	l hot			%	
g	level				

4 CONCLUSION

In this paper, 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 tuning 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 Genaro Motti, "Machine Learning in the Support of Context-Aware Adaptation", Université catholique de Louvain, 2017.
- [16] Christos Stergiou and Dimitrios Siganos, "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-Eun Ko, "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-tao Zheng, 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 April 6, 2019.
- [35] WouterGevaert and GeorgiTsarov, "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.L. Hubert and P. Arabie, "Comparing Partitions," *J. Classification*, vol. 2, no. 4, pp. 193-218, Apr. 1985. (Journal or magazine citation)
- [40] R.J. Vidmar, "On the Use of Atmospheric Plasmas as Electromagnetic Reflectors," *IEEE Trans. Plasma Science*, vol. 21, no. 3, pp. 876-880, available at <http://www.halcyon.com/pub/journals/21ps03-vidmar>, Aug. 1992. (URL for Transaction, journal, or magazine)
- [41] J.M.P. Martinez, R.B. Llavori, M.J.A. Cabo, and T.B. Pedersen, "Integrating Data Warehouses with Web Data: A Survey," *IEEE Trans. Knowledge and Data Eng.*, preprint, 21 Dec. 2007, doi:10.1109/TKDE.2007.190746.(PrePrint)

Author: Abdi Mosisa is graduated with MSc in Computer Science in 2020 from Addis Ababa University and BSc in Computer Science in 2017 from Ambo University. He is employed at Arsi University since October 2017. His current research interests are machine learning, neural network, internet of things and embedded systems.