CHAPTER 4

NEURAL NETWORK BASED SENSOR FAULT IDENTIFICATION IN NETWORK CONTROL SYSTEM

4.1 INTRODUCTION

In a real time control systems, the acquisition of information from the physical environment can be obtained from sensors and whose output data can be transmitted over a communication network to the controller for generating the required control action. This control information transfer suffers from various limitations like environmental factor, lack of processor storage memory, constrained communication resources and so on. The stability of the system control depends on the reliable data transfer from sensor to the controller and from controller to the actuator. Any malfunction in the sensor output results in the transmission of faulty data to the controller, hence results in the loss of system stability. In NCS, a failure model can be specified for sensing, control processes, actuating process and communication channels. Usually, it is considered that communication channels can delay or loose messages and that processes can exhibit the faulty behavior such as crash failures, omission failures, timing failures, and arbitrary failures. The challenge here is how to monitor and diagnose the sensor performance under abnormal conditions to make the sensor output reliable and to maintain the accuracy of the sensor data and quality of service.

The traditional and simple way of identifying failure of sensor is by using redundant sensors for the same measurement process. This further
increases the complexity of the system and not providing any mechanism to locate the failure sensor. The necessity for neural network is highly appreciable under such conditions, since it provides different methods to train the actual input to map with the desired output. Using neural network, failure identification and correction easily achieved with a small increase in cost of the memory.

This chapter discusses the concept of Recurrent Neural Network (RNN) with layer feedback method to train the sensor data to map the input to output relationship. Kalman filtering technique is used to estimate the error and also to provide desired data as the output under sensor failure.

4.2 OVERVIEW OF NEURAL NETWORK

Neural Network (NN) is a technique that performs computation similar to human brain to achieve good performance. NN is a massively parallel distributed computation technique made up of simple processing units or neurons, which has a natural propensity for storing experimental knowledge and making it available for use. It resembles brain in two respects:

1. Knowledge is acquired by the neural from its environment through a learning process.
2. Interneuron connection strengths, known as synaptic weights, are used to store acquired knowledge.

Figure 4.1 Diagram of a Neuron
NN based models derive their structure as a model of living neurons. Figure 4.1 represents a typical neuron and Figure 4.2 shows a typical neural network element that emulates the neuron. It receives input $X$ from different sources and has an associated weight $W_{ji}$, computes an activation function. Its output $Y_i$, in turn, can serve as input to other units.

A Recurrent Neural Network (RNN) is defined as one in which either the network's hidden unit activations or output values are fed back into the network as inputs. RNNs are mathematical abstractions of biological nervous systems that can perform complex mappings from input sequences to...
output sequences. Unlike traditional, programmed computers, RNNs learn their behavior from a training set of correct example sequences. As training sequences are fed to the network, the error between the actual and desired network output is minimized using gradient descent, whereby the connection weights are gradually adjusted in the direction that reduces this error most rapidly. There are two basic ways of applying feedback to the neural network: local feedback at the level of single neuron inside the network, global feedback encompassing the whole network. The application of feedback enables recurrent network to acquire state representation, which make them suitable diverse application as nonlinear prediction and modeling, speech processing and automobile engine diagnostics.

The use of global feedback has the potential of reducing the memory requirement significantly. The common features of the recurrent network architecture are, it incorporates a static multilayer perceptron and exploit the nonlinear mapping capability of the multilayer perceptron. The fully connected recurrent network is a state space model in which only those neurons in the multilayer perceptron that feedback their output to the input layer with delays are responsible for defining the state of the recurrent network. An important property of a recurrent network described by state space model is that it can approximate a wide class of nonlinear dynamic system. The fully connected recurrent network consists of say q neurons with m external inputs as shown in Figure 4.4. The network has two distinct layers: a concatenated input feedback layer and a processing layer of computation nodes. Correspondingly, the synaptic connections of the network are made up of feedforward and feedback connections.
The synaptic weights of neuron in the hidden layer are connected to the feedback nodes in the input layer. Similarly, the synaptic weights of the hidden neurons are connected to the source node in the input layer and the synaptic weight of the linear neurons in the output layer that are connected to the hidden neurons. The synaptic weights of neurons in each layer are used to form a matrix called as weight matrix. It is assumed that the bias terms for the hidden neurons are absorbed in the weight matrix.
4.2.1 Learning Process

The learning process or rule is a procedure for modifying the weights and biases of a network to train the network to perform some task. There are many learning rules for NN namely supervised learning, unsupervised learning and reinforcement or graded learning.

In supervised learning, the learning rule is provided with a set of examples, called as training set, of proper network behaviour:

\[ \{ p_1, t_1 \}, \{ p_2, t_2 \}, \ldots, \{ p_q, t_q \} \]

where \( p_q \) is an input to the network and \( t_q \) is the corresponding correct output. As the input set is applied to the network, the network outputs are compared to the targets. The learning rule is then used to adjust the weights and biases of the targets.

Reinforcement learning is similar to supervised learning, except that, instead of being provided with the correct output for each network input, the algorithm is only given a grade. The grade is a measure of the network performance over some sequence of inputs. This type of learning is currently much less common than supervised learning.

In unsupervised learning the network is autonomous: it just looks at the data it is presented with, finds out about some of the properties of the data set and learns to reflect these properties in its output. What exactly these properties are, that the network can learn to recognise, depends on the particular network model and learning method. There are two modes of training an ordinary (static) multilayer perceptron: batch mode and sequential mode. In batch mode, the sensitivity of the network is computed for the entire
training set before adjusting the free parameters of the network. In the sequential mode, parameter adjustments are made after the presentation of each pattern in the training set. Likewise there are two modes of training in recurrent network.

1. **Epochwise training:** For a given epoch, recurrent network starts running from some initial state until it reaches a new state, at which point the training is stopped and the network is reset to an initial state for the next epoch.

2. **Continuous training:** This method is suitable for situations were there are no reset states available and on-line learning is required. The distinguishing feature of continuous learning is that the network learns while signal processing is being performed by the network. Here the learning process never stops.

Each time a pattern is presented, the unit computes its activation just as in a feed forward network. However its network input now contains a term which reflects the state of the network (the hidden unit activation) before the pattern was seen. For the present subsequent patterns, the hidden and output units will be a function of everything the network has seen so far.

There are different optimization methods used in neural network learning like adjoint, Markov chain, Monte Carlo and genetic algorithm. These methods differs in how they locate the minimum (gradient-descent or global search), how observations are processed (concurrent or sequentially), or the number of iterations used, or assumptions about the statistics. The different methods are equally successful at estimating the parameters however, the biggest variation in parameter estimates arise from the tuning of
cost function. Relatively poor results are obtained when the model-data mismatch in the cost function include weights that are instantaneously dependent on noisy observations. Missing data cause estimates to be more scattered, and the uncertainty of predictions increase correspondingly. All methods give biased results when the noise is correlated or non-Gaussian, or when incorrect model forcing is used.

### 4.2.2 Levenberg-Marquardt Learning

The Levenberg-Marquardt (LM) method is a gradient-descent method that combines information about the gradient and its second derivative to efficiently locate the minimum. Gradient descent is simply a technique in which parameters, such as weights and biases, are moved in the opposite direction of the error gradient. Each step down the gradient results in a smaller error until a minimum error is reached. The use of the momentum term changes the algorithm slightly by making the parameter changes proportional to a running average of the gradient (Hagan and Menhaj, 1994). The LM algorithm possesses quadratic convergence when it is in the vicinity of (but not too close to) a minimum.

LM uses gradient descent to improve on an initial guess for its parameters and transforms to the Gauss-Newton (GN) method as it approaches the minimum value of the cost function. Once it approaches the minimum, it transforms back to the gradient descent algorithm to improve the accuracy. Due to its desirable convergence capabilities, in many optimization applications, the LM method is usually preferred over many other optimization techniques. Despite all of its popularity in optimization society, the LM technique was not adopted for training neural networks until recently, due to its complexity. Although the application of the LM algorithm to NN
learning is still in the early stages, its encouraging results have led to its use throughout the remainder of this study.

**4.2.3 Real Time Recurrent Learning**

Real Time Recurrent Learning (RTRL) is based on gradient decent uses the minimum amount of available information, namely an instantaneous estimate of the gradient of the cost function with respect to the parameter vector to be adjusted. The learning process can be accelerated by exploiting Kalman filter theory, which utilizes information contained in the training data more efficiently. In real time learning, the adjustments are made to the synaptic weight of a fully connected network in real time.

The Kalman filter is a sequential technique for state estimation and in this chapter the Extended Kalman Filter (EKF) is used with parameters included in the state vector. Both LM and EKF take few hundred iterations. The EKF requires the Jacobian for the model, which may be difficult to specify for a complicated model, and would need to be updated if the model code changed (Straub and Shroder, 1996). The LM method is available in easy-to-use packages copes easily with changes to model code, but is expensive when many parameters are estimated simultaneously. All methods involve some choices, such as stopping criteria, or specification of model error for the Kalman filter. Another issue is the number of parameters that can be inverted simultaneously by the different methods. A high number of parameters may be a problem for down-gradient approaches like LM, depending on the model and whether there are multiple minima.

The EKF gives better performance when compared with batch methods, for strongly nonlinear models. The Kalman filter starts with the initial guess for the state variables and updates their estimates sequentially.
The state variable estimates at any time are due to observations up to and including that time. The parameter estimates from the final time are taken as the best estimates, because they have been influenced by all observations. A particular strength of the Kalman filter is the ability to assimilate observations allowing for errors in the forcing or the model. The EKF involves linearization of the model equations about the current trajectory and does estimation of parameters by including them in the state vector. This choice was based on experience with the EKF that has shown better results for parameter estimation when greater uncertainties are used than would be suggested by residuals. Increasing the observation error gave better results for parameter estimation than increasing the model error.

A Kalman state estimator requires a state-space model of the plant, the process and measurement noise covariance data. Continuous learning based on gradient decent, is typically slow due to reliance on instantaneous estimates of gradients. This serious limitation can be overcome by viewing the supervised training of a recurrent network as an optimum filtering problem, the solution of which recursively utilizes information contained in the training data in a manner going back to the first iteration of the learning process.

Also the theory is formulated in terms of state space concepts, providing efficient utilization of the information contained in the input data. Estimation of the state is computed recursively, that is, each update estimate of the state is computed from the previous estimate and the data currently available, so only the previous estimate require storage. The superior learning performance of the EKF (hereafter referred simply as Kalman filter) over back propagation is due to its information-preserving property. Thus the complete RNN based sensor model during the learning process is shown in Figure 4.5.
4.3 SENSOR MODELLING

By system it is considered the actual underlying physics that generates the data, whereas by model we consider the mathematical model of the system. There are three classes of input-output modeling technique they are:

1. Parametric modeling assumes a fixed structure for the model. The model identification problem then simplifies to estimating a finite set of parameter for this model. This estimation is based upon the prediction of real input data, so as to best match the input data dynamics.

2. Nonparametric modeling seeks a particular model structure form the input data. The actual model is not known beforehand.
3. Semi parametric modeling is the combination of the above two.

To understand and analyse the real world physical phenomena, various models have been considered (Younghwan An, 1998). Depending on some priori knowledge about the process data and model, there are three general types of modeling.

1. White box modeling in which given data gathered from plant dynamics provide an initial framework in building mathematical model of the process.

2. Black box modeling assumes no previous knowledge about the system that produces the data. The aim hereby is to find the function F that approximates the process y based on previous observation of the process y_{past} and input u.

3. Gray box model is the natural compromise between the two previous models.

The exact form of input-output model that describes a real-world system is most commonly unknown. The black box model establish a functional dependency between the input and output which can be either linear or non linear. Once the training process is completed the neural network represents a black box, nonparametric process model.

NN models may be used for process models and are suited for handling non-linear systems. Where an analytical model is not available, NN is the experimental method for the general non-linear modeling process. Dynamic processes may also be modeled by including inputs with different times. This allows the model to emulate a difference equation. The output
signal of a nonlinear model can be considered as a combination of outputs from suitable sub models.

The structure identification, model validation and parameter estimation based upon these sub models are more convenient than those of the whole model. Block oriented stochastic models consist of static nonlinear and dynamic linear modules. Such model often occur in practice, example of which are

1. The Hammerstein model, where a zero-memory nonlinearity is followed by a linear dynamic system characterised by its transfer function $H(z) = N(z)/D(z)$.

2. The Wiener model, where a linear dynamical system is followed by a zero-memory nonlinearity.

The definition of certain stochastic models are given by the

1. **Wiener system**
   \[ y(k) = g(H(z^{-1})u(k)) \] (4.1)
   where $u(k)$ is the input to the system and $y(k)$ is the output,
   \[ H(z^{-1}) = C(z^{-1})/D(z^{-1}) \] (4.2)

   Equation (4.2) is the $z$-domain transfer function linear component of the system and $g(.)$ is a nonlinear function;

2. **Hammerstein system**
   \[ y(k) = H(z^{-1})g(u(k)) \] (4.3)

3. **Uryson system,** defined by
   \[ y(k) = \sum_{i=1} H_i(z^{-1})g_i(u(k)) \] (4.4)

A sensor is truly a dynamic system, and thus a good understanding of sensor performance requires the understanding of its dynamic response.
The dynamics of a sensor are relevant when the sensor input is a time-dependent signal. On the other hand, many applications involve sensor inputs that are constant over long time periods. The dynamic structure of a sensor can have almost any form, the most general being

\[
q' = f(q, x) \quad (4.5) \\
V = g(q, x) \quad (4.6)
\]

where \( q \) is the internal sensor state (possibly a vector), \( x \) is the physical input to the sensor and \( V \) is the sensor output. The feedback effects due to non-ideal impedances (or their analogue for the particular application) are assumed to be included in the block-structured Hammerstein-Wiener nonlinear feedback dynamic sensor model has a linear dynamic block surrounded by three static nonlinear blocks.

![Figure 4.6 Hammerstein-Wiener nonlinear feedback dynamic sensor model](image)

4.3.1 Sensor Failure Types

A generic sensor failure can be modeled as:

1. **Additive type sensor failure:** This sensor failure can be modeled by adding a constant bias to a sensor’s nominal value. The additive bias sensor failure can be expressed as follows,

\[
X_{\text{failure},i} = X_{\text{nom},i} + \rho n_i \quad (4.7)
\]
where $n_i$ is the direction vector for the $i^{th}$ faulty sensor, and $\rho$ is the magnitude of the failure which can be positive or negative.

2. **Multiplicative type sensor failure:** This sensor failure can be modeled by multiplying a factor to a sensor’s nominal value.

\[
X_{\text{failure},i} = (1.0 + k n_i )X_{\text{nom},i}
\]  

(4.8)

where $k$ is an amplitude for the $i^{th}$ faulty sensor. According to the direction vector, $n_i$, sensor failure can be modeled as step-type sensor failures, $n_i = 1$.

3. **Stuck with constant bias sensor failure:** This sensor failure can be modeled by setting measurement from the failed sensor $n_i$ to a constant bias $\rho$. The stuck with constant bias sensor failure then can be described as follows,

\[
X_{\text{failure},i} = \rho n_i
\]

(4.9)

Additionally, the types of sensor failure mentioned above may become temporary non-operational (intermittent failures).

4.4 **RNN FOR SENSOR FAULT TOLERANCE**

RNN are used here to model a sensor node, its dynamics and fault detector. Considering large number of sensors in NCS having their own dynamics, they interact between themselves via the communication network and the base station which controls the network. To dynamically model such sensors, without a loss of generality, it assumes that there is one sensor per sensor node. More sensors per node will just increase the size of the RNNs.
Figure 4.7 Topology of Networked Sensor with NN

Figure 4.8 Flow chart for fault identification and correction in sensor #2
Sensor nodes can be viewed as small dynamic systems with memory-like features. Output of one node forwards the information to the next node. NCS with large number of sensors embedded with NN are interconnected through a communication network is shown in Figure 4.7. The effect of using communication network in NCS is modeled by using confidence factors (cf), whose values can be in the range of $0 < \text{cf} < 1$ between the sensor nodes. It is assumed that communication links are symmetric, i.e., if the NN of the sensor node 1 communicates with the NN of the sensor node 2, then opposite is also true with the same confidence factor. The confidence factor depends on the signal strength and data quality in communication links between the nodes. Note that the confidence factors do not provide stochastic modeling of the communication channel. The overall RNN model of a sensor network for fault tolerance is shown in the Figure 4.8.

### 4.4.1 Adaptive Threshold

The difference between sensor output and RNN output is used to identify the fault. The sensor is said to be fault if this difference exceeds the threshold value. To reduce the delay in fault identification and to avoid false alarm adaptive threshold method is used (Perhinschi M.G. et al, 2006).

![Floating Limiter Diagram](image-url)
The floating limiter concept is shown in Figure 4.9 is used to compute variable thresholds for the parameters involved in the fault identification process. The floating limiter has some unique features, which allow full authority and rates within a domain, but rate limits if the input signal persists in one direction, with the objective of preventing a hard over from propagating into the control laws. The floating limiter is designed to fluctuate around the input signal and drift with it at a rate equal to the rate of the signal but less than an imposed limit. High rates of change are permitted only within floating limiter upper and lower bounds. The upper bound $X_{UB}$ at time step $n$ is defined as

$$X_{UB}(n) = \min (X(n) + \Delta(X) \cdot X_{UB}(n-1) + R_{\text{max}}(X) \cdot [t(n) - t(n-1)])$$

The lower bound $X_{LB}$ at time step $n$ is then defined as

$$X_{LB}(n) = \max (X(n) + \Delta(X) \cdot X_{LB}(n-1) + R_{\text{max}}(X) \cdot [t(n) - t(n-1)])$$

where $\Delta$ represents the range within which the signal $X$ is allowed to vary at any rate and $R_{\text{max}}$ represents the maximum rate at which the bounds are allowed to vary. Both these parameters depend on the nature of the input signal but are constant in time. As soon as boundary is reached, the monitored signal is limited to the value reached on the bound and additional safety measures may be taken.

The threshold imposed on parameter $X$ is given by the following relationship based on the statistics of $X$.

$$X_{(\text{Thd})} = \bar{X} + \beta \cdot \sigma(X) + b$$

(4.10)

where $\bar{X}$ is the average value of $X$, $\sigma(X)$ is the standard deviation of $X$, $\beta$ is a bound factor and $b$ is a bias. The statistic parameters are computed over a window of width $\delta$. The three design parameters involved ($\beta$, $b$ and $\delta$) are
constant in time but may depend on the nature of the input signal. Approximations of the average and the standard deviation over a time window of width $\delta = 1$ are computed using the following autoregressive moving average filter

$$D(z) = 0.025(z^3+z^2+z+1)/(z^3-0.3z^2-0.3z-0.3) \quad (4.11)$$

The relationship is then used to compute the adaptive threshold using the parameters $\beta$ and $b$.

### 4.5 MATLAB SIMULATION RESULTS

Here the sensors with two neighboring nodes are considered for simulation. Sensor is dynamically modeled using Hammerstein – Wienner non linear feedback dynamic sensor using $\text{arctan}(.)$ in nonlinear part and dynamic element as $H(s) = 1/(s^2+s+1)$. The output of the sensor, present and past output of neighboring sensor are given to RNN and trained using back propagation function for 100 maximum epochs. Neural network activation function is chosen as a standard sigmoid function.

![MATLAB/SIMULINK block for sensor model](image)

**Figure 4.10** MATLAB/SIMULINK block for sensor model
Input to the sensor is considered as
\[ U_i(k) = 10 + \sin((i+2\pi k)/3) + n_i(t) \]  \hspace{1cm} (4.12)

where \( n_i(t) \) is the white noise of variance 0.6 at sensor node as shown in Figure 4.11.

Figure 4.12 shows the Mean Square Error (MSE) of the desired output and the sensor output for the training of 100 epoch. The MSE reached is 0.000579 and MSE with Kalman (MSEk) is 0.000034 for RNN learning with the confidence factor set to 1.
The plot shown in Figure 4.13 compares the desired output with the response of fully connected RNN trained without using the Kalman filter algorithm and with setting the confidence factor equal to 1.

![Figure 4.13 RNN response without Kalman filter algorithm and cf=1](image)

The plot as shown in Figure 4.14 compares the desired output with the response of fully connected RNN trained with Kalman filter algorithm when confidence factor set to 1.

![Figure 4.14 RNN response with kalman filter algorithm and cf=1](image)
From Figure 4.12, Figure 4.13 and Figure 4.14, the RNN response closely tracks the desired output when the algorithm includes the Kalman filter, as it takes fewer epochs to train when compared to RNN response without using Kalman filter. The plot as shown in Figure 4.15 compares the desired output with the response of the fully connected RNN trained without Kalman filter algorithm with confidence factor equal to 0.8 (i.e., noisy signal).

![Figure 4.15 RNN response without kalman filter algorithm and cf=0.8](image)

The plot shown in Figure 4.16 compares the desired output with the response of fully connected RNN trained with kalman filter algorithm with confidence factor equal to 0.8. The performance of Kalman filter algorithm is better even under the disturbance condition.
Figure 4.16 RNN response with kalman filter algorithm and cf=0.8

Figure 4.17 RNN response with kalman filter algorithm for untrained input set
Figure 4.18 RNN response without kalman filter algorithm for untrained input set

From Figure 4.17 and 4.18, it is shown that the RNN trains in better manner for untrained input set even in case of a noisy signal by using Kalman filter when compare to without Kalman filter. Hence the network generalizes well for Kalman filter learning as shown in Table 4.1.

### Table 4.1 Comparison of Mean Square Error for with and without kalman

<table>
<thead>
<tr>
<th></th>
<th>With kalman</th>
<th>Without kalman</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cf = 1</td>
<td>0.000034</td>
<td>0.000579</td>
</tr>
<tr>
<td>Cf = 0.8</td>
<td>0.000051</td>
<td>0.001174</td>
</tr>
<tr>
<td>Untrained input set</td>
<td>0.000007</td>
<td>0.000001</td>
</tr>
</tbody>
</table>

The plot in Figure 4.19 shows the response of the sensor for stuck at fault generated based on the equation (4.9) with $\rho n_i = 1$, and Figure 4.20 shows the error at 70th time unit for stuck at fault using RNN with kalman filter algorithm with adaptive threshold. For the adaptive threshold approach, the ratio can be tuned mainly by varying the bound factor and the bias in
The parameters of the adaptive threshold scheme have been selected based upon a limited number of cases where the constant threshold approach has failed. Also, for better performance bound factor $\beta = 2$ and bias $b= 0.00001$ is used in all the further cases. In case of adaptive threshold, the declaration of fault detection will be accurate, however it takes few sample time.

**Figure 4.19** RNN with kalman response for the sensor with stuck at fault

**Figure 4.20** Adaptive Threshold for stuck at sensor fault-RNN with kalman
The plot as shown in Figure 4.21 shows the response of the sensor for step type fault generated based on the equation (4.8) with $k = 0.1$, and Figure 4.22 shows the error at 70th time unit for step type fault using RNN with kalman filter algorithm with adaptive threshold.

![Figure 4.21 RNN with kalman response for the sensor with step type fault](image1)

![Figure 4.22 Adaptive Threshold for step type sensor fault –RNN with kalman](image2)
Figure 4.23 RNN with kalman response for the sensor with additive fault

Figure 4.24 Adaptive Threshold for additive sensor fault-RNN with kalman
The plot as shown in Figure 4.23 shows the response of the sensor for additive fault generated based on equation (4.7) with $\rho n_i = -1$, and Figure 4.24 shows the error at 70\textsuperscript{th} time unit for additive fault using RNN with kalman filter algorithm with adaptive threshold.

![Figure 4.25](image_url)

**Figure 4.25** RNN without kalman response for the sensor with step type fault

![Figure 4.26](image_url)

**Figure 4.26** Adaptive Threshold for step type sensor fault – RNN without Kalman
The plot as shown in Figure 4.25 shows the response of the sensor for stuck at fault generated based on the equation (4.9) with \( \rho n_i = 1 \) and Figure 4.26 shows the error at 70th time unit for stuck at fault using RNN without Kalman filter algorithm with adaptive threshold.

![Figure 4.27 RNN without Kalman response for the sensor with stuck at fault](image)

![Figure 4.28 Adaptive Threshold for stuck at sensor fault - RNN with kalman](image)
The plot as shown in Figure 4.27 shows the response of the sensor for step type fault generated based on the equation (4.8) with $k = 0.1$ and Figure 4.28 shows the error at 70th time unit for step type fault using RNN without kalman filter algorithm with adaptive threshold.

Figure 4.29  RNN without Kalman response for the sensor with additive fault

Figure 4.30  Adaptive Threshold for additive sensor fault-RNN without Kalman
Figure 4.29 shows the response of the sensor for additive fault generated based on the equation (4.7) with $\rho_{n_i} = -1$, and Figure 4.30 shows the error at 70th time unit for additive fault using RNN without Kalman filter with adaptive threshold.

Comparing the three different fault characteristics of adaptive threshold with Kalman and without Kalman, the characteristics are seems to be having similar resemblance. However the Kalman algorithm takes less time to detect the fault. In case of adaptive threshold, how much samples are required for fault declaration depends on the application.

4.6 FPGA IMPLEMENTATION OF RECURRENT NEURAL NETWORK

Recurrent neural network can be implemented either using microcontrollers or FPGA. FPGA based NN is advantageous as it paves role for design and building of reconfigurable devices. A hardware design implementation of sensor’s RNN model by using FPGA is presented. In this section the RNN modeling is to obtain input–output data sets to use for RNN model training using sensor model discussed in section 4.2. In second step, ANN model of sensor is modeled by using software such as Matlab. For the RNN model of sensor, the biases and weights are to be evaluated in offline. RNN parameters were used to model the VHDL program
4.6.1 Architectural model of the neural network

Requirements of ANN hardware architecture are parallelism, flexibility and their relationship to the number of logic blocks. Parallelism is achieved by means of at least one multiplier per layer.

The FPGA equivalent architectural model of the neuron is shown in Figure 4.32. The hardware model is mainly based on:

- Memory circuit (ROM) where the final values of the weights are stored
- Multiply accumulate circuit (MAC) which computes the weighted sum
- Look up table (LUT) which implements the sigmoid activation function.
ANN architecture has the following features:

- For the same neuron, only one MAC is used to compute the product sum
- Each MAC has its own ROM of weights whereby the depth of each ROM is equal to the number of nodes constituting its input layer
- For the same layer, neurons are computed in parallel.
- Computation between layers shall be done serially
- The whole network shall be controlled by a unit control

The artificial neural network modeled is a three layered recurrent network in which the input layer acts as a buffer for the inputs. It consists of an input layer with 2 nodes, a first hidden layer with 5 nodes and an output layer with 1 node. Depending on the network type to be realized, different non-linear activation functions for neurons are required. Some models need no more than a simple hard limiter threshold function. In other models, more complex nonlinearities are involved like the sigmoid function.

A straightforward approach to obtain these functions is using circuits for full calculations that are multipliers, dividers, adders, comparators and others. However, this may be a very time consuming and area consuming approach. To avoid unnecessary computations, a very common method is to use look-up tables. The whole neural network structure is made up of an array of nodes or neural elements. These nodes do the important computations and they are interconnected with other nodes in the adjacent layer. Thus there are two kinds of approach that can be adopted to map the neural models onto array architectures.
They are:

1. The direct design approach, which maps structure of a neural model directly into hardware
2. The indirect approach, which exploits the matrix processing nature of neural models to simplify the hardware requirement considerably.

In this chapter, the first approach is adopted. The multiplication of constant weights inside each node can be implemented by two different algorithms: (1) the ROM LUT and adders, (2) multipliers.

The design and implementation of neural networks are based on arrays that hold the node values and connection weights. This approach models the conceptual network more closely and makes the VHDL code more understandable and maintainable. A node sums all its inputs and passes the result through the transfer function to obtain its output. The output of a node passes over connections to become the input of other nodes. A constant weight is associated with each connection, and the value passing over the connection is multiplied by the weight of the connection to obtain the input value delivered to the receiving node.

The nodes in the input layer do not receive input from other nodes and do not pass values through a transfer function. They simply act as a buffer, that is, they distribute input values from an external source over connections to other nodes. Output nodes do not pass their output values over connections to other nodes but provide values directly to an external link. Between the input and output nodes are the hidden nodes, which obtain their inputs from other nodes, and distribute their outputs over connections to other nodes. A neural network with no hidden nodes, in which the input nodes
connect directly to the output nodes, is called a perceptron. Object-based design encourages an implementation to model its variables on the objects such as the nodes in the neural network, to encapsulate the types defining these objects with the operations on them.

![FPGA modeling of neural network](image)

**Figure 4.32 FPGA modeling of neural network**

### 4.6.2 MODELSIM Outputs

Target device considered for simulating the RNN model for sensor fault identification is XC2S50E6PQ208. The VHDL code for RNN with two input and five neuron in the hidden layer is simulated using Modelsim software and the output is shown in Figure 4.33. This output is a logical output pattern of RNN for the given input set.
The VHDL model is not synthesizable if the declaration of input and output variable is declared as real data type. To develop synthesizable code, floating point arithmetic can be used. For simplicity an assumption is made such that only mantissa parts of fixed exponent part are processed. For example all inputs are given in the power of \((e^{-4})\) and in addition the exponent part will not change, however, in multiplication the exponent part will become the power of \((e^{-8})\) but only the four most significance digits are used for further computation.

![MODELSIM wave form output of RNN](image)

**Figure 4.33** MODELSIM wave form output of RNN

![RTL Schematic block view of RNN](image)

**Figure 4.34** RTL Schematic block view of RNN
In Figure 4.34, B is the bias I is the sensor output and X1, X2 are the output of neighboring sensor. Y is the RNN output. The weights of RNN trained one obtained from the MATLAB simulation; hence only off-line training is used i.e. RNN is trained well and then put in to practice. If the flag bit is set to logic one then it indicates the presence of fault between sensor output I and RNN output Y. The level of fault tolerance is 0.1.

Figure 4.35  MODELSIM synthesizable signal output of RNN
Figure 4.36 MODELSIM synthesizable wave form output of RNN

Figure 4.36 shows all inputs to the RNN, layer output and the final RNN output. The fault condition is identified using the flag bit. The flag bit is set to 1 at 600 ns which indicates the presence of sensor fault.

4.6.3 Synthesis Report

The synthesisization is done by using FPGA technology to determine the implementation factors like memory usage, execution timing and many. The device used for synthesisization is XC2S50E6PQ208. The speed is set in the standard mode, the product is set for commercial applications and the synthesis is done for the worst case. Table 4.2 gives cell usage and Table 4.3 gives the Timing and Memory usage.
Table 4.2  Cell Usage

<table>
<thead>
<tr>
<th>Cell</th>
<th>Cell usage</th>
<th>Cell</th>
<th>Cell usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>BELS</td>
<td>43926</td>
<td>LUT1</td>
<td>2682</td>
</tr>
<tr>
<td>BUF</td>
<td>157</td>
<td>LUT2</td>
<td>7942</td>
</tr>
<tr>
<td>GND</td>
<td>1</td>
<td>LUT3</td>
<td>1677</td>
</tr>
<tr>
<td>INV</td>
<td>3399</td>
<td>LUT4</td>
<td>3669</td>
</tr>
<tr>
<td>MULT_AND</td>
<td>7</td>
<td>IBUF</td>
<td>128</td>
</tr>
<tr>
<td>VCC</td>
<td>1</td>
<td>OBUF</td>
<td>33</td>
</tr>
<tr>
<td>FlipFlops/Latches</td>
<td>183</td>
<td>Shift Registers</td>
<td>25</td>
</tr>
</tbody>
</table>

Table 4.3  Timing and Memory Summary

<table>
<thead>
<tr>
<th>Minimum period</th>
<th>4.666ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum input arrival time before clock</td>
<td>80.789ns</td>
</tr>
<tr>
<td>Maximum output required time after clock</td>
<td>47.194ns</td>
</tr>
<tr>
<td>Maximum combinational path delay</td>
<td>35.264ns</td>
</tr>
<tr>
<td>Total memory usage</td>
<td>416896 kb</td>
</tr>
</tbody>
</table>

4.7  CONTRIBUTION OF THIS CHAPTER

The contribution of this chapter is, the fully connected RNN based sensor model is constructed to prevent the transmission of faulty data which will affect the stability of the system. The various fault cases are considered and the performance of fully connected RNN is evaluated by employing two different training algorithm viz., LM and EKF. It is observed that the EKF tracks the desired response and takes less number of epochs to identify the fault. The use of adaptive threshold in fault identification process is to provide protection against malfunction of other NCS components by limiting signals to values on variable upper or lower bounds. FPGA modeling of RNN based sensor fault identification is done and verified the logic for a single fault in a sensor.