UNIVERSITY OF NAIROBI

Using Neural Networks to reduce noise in Internet of Things data streams

Samuel Magondu - P53/79707/2015

Supervisor: Prof Peter Waiganjo Wagacha

November 2016

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Distributed Computing Technology in the School of Computing and Informatics of the University of Nairobi

ABSTRACT

Noise in the Internet of Things is threatening to drown out sensor data. The problem is growing as more and more devices are being connected to the internet. The noise comes from the electric components both within and without the IoT devices. Other sources of noise include poor calibration. There is thus a need to ensure accurate data is collected in a cost effective way as noisy data might prove disastrous.

This study sought to find out the suitability of using neural networks as a filter and also compared its performance to a Kalman filter. An Artificial Neural Network filter application was developed using rapid application prototyping using simulated data to test. The results showed that the Artificial Neural Network filter was reliable to filter out the noise compared to other filtering solutions such as the Kalman filter. Despite the Artificial Neural Network being about 15 times slower than the Kalman filter, it was found to be more accurate. It was thus found that an Artificial Neural Network is much more accurate than a Kalman filter and makes a good noise filter for IoT devices.

DECLARATION

I, Samuel Magondu Njenga, do hereby declare that this research project is entirely my own work and where there is work or contributions of other individuals, it has been duly acknowledged.

To the best of my knowledge, similar research work has not been carried out before or previously presented to any other educational institutions in the world of similar purposes or form.

Sign: ---------------------------------- Date: ------------------------------------

Name: Samuel Magondu Njenga

Reg.No. P53/79707/2015

Supervised By:

This project has been submitted in partial fulfillment of the requirement for the Master of Science degree in Distributed Computing Technology of the University of Nairobi with my approval as the University supervisor.

Sign: -------------------------------- Date: ------------------------------------

Prof. Peter Waiganjo Wagacha

DEDICATION

To my fiancée, Eunice Wairimu Kamau

Parents, Peter Njenga and Margaret Njeri

and

My siblings, Sarah Wairimu and Eliud Ngorongo

ACKNOWLEDGEMENT

First, I wish to thank God for his grace and mercies for without him, I wouldn’t have made it this far.

Secondly, I would like to relay my sincere gratitude to my supervisor, Prof Peter Waiganjo Wagacha and panelists, Dr. Samuel Ruhiu, Dr. Daniel Orwa and Mr. Christopher Moturi for their guidance, positive feedback and above all their valuable time and advice throughout my project work.

I would also like to thank the team at Upande Ltd for their technical assistance throughout this project. They were helpful in providing advice as well resources needed to make this project possible.

Finally, I wish to express my appreciation to my parents Peter and Margaret for the sacrifice they made for the sake of my education. They have been the shoulders of giants I have stood on all through my education, my fiancée Eunice for her unconditional support, prayers, and encouragement throughout my study. Without them, this project would not be successful.

TABLE OF CONTENT

Contents

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

ANN	Artificial Neural Networks
API	Application Programming Interface
CASE	Computer Aided Software Engineering
GPS	Geographical Positioning System
IoT	Internet of Things
JSON	JavaScript Object Notation
RAD	Rapid Application Development
RDBMS	Relational DataBase Management System
RFID	Radio Frequency Identification
UI	User interface

CHAPTER 1: INTRODUCTION

Internet of Things (IoT) is changing the way we do things. Now more than ever, we have cheap sensors. In addition, the internet is now pervasive and even basic devices such as television sets can connect to the internet. This of course, poses a huge challenge in regards to the data produced. The challenge is in the amount and quality of the data produced. Words like "Big Data" often associated with IoT because of the streams of data flowing in.

Data collected via IoT has a sizable amount of noise. McHenry, et.al, 2015 go to an extent to claim that IoT is drowning under radio frequency noise. Electronic devices near IoT devices are likely to introduce noise and skew the reading. However, noise is not only introduced by electronic devices. Man-made activities can make the data noisy such as resetting devices etc. Another source of noise in IoT devices is poor calibration. Cheap devices are often fitted with sensors that have high error rates that contribute to errors in the data collected. The cost is often associated with the quality of the device. As such, users collecting data may be forced to use low cost IoT devices in an effort to save on costs which might lead to collection of erroneous data.

The noise in the data can be reduced using various methods and approaches. Filtering algorithms can be used as well as introducing intelligent filters in the IoT devices. Sometimes, a combination of both is used to improve on performance. Algorithms can be used to train intelligent applications on what noisy data is thus replacing algorithms that might be resource intensive. Neural networks can be trained and used as filters making such intelligent applications.

1.1 Background

Sensors are often placed in many devices we use daily. They are embedded in our environment to collect data that is crucial in decision making. Given they are placed in the environment, they can be subject to harsh conditions that impact their accuracy. For example, a temperature sensor could easily be affected by passing objects that emit heat thus skewing their reading. This introduces noise in the data which can be catastrophic depending on the kind of system the data is used in. The problem if further compounded by the amount of devices out there. According to (Frenzel, 2016), there will be about tens of billions of devices deployed in a few years to come according to projections. The deployed devices will also interfere with nearby devices introducing noise. Whereas this is good news for the IoT community as a whole, it is bad news for those looking to collect accurate data.

Noise filtering algorithms are available and can easily be programmed into the sensors. However, many sensors have limited computing resources (memory, processing capabilities) which might hamper the efficiency of such algorithms. Another design goal when creating sensors is to make sure they are energy efficient and thus, the designer has to make decisions regarding which tools to use to save on energy. IoT sensors also have to work with limited bandwidth and connectivity thus making it crucial to ensure the sanctity of the data.

As such, any engineer has to find a way to reduce the noise while taking into consideration the resources available on the sensors. The usage of sensors is growing and domain specific solutions are not scalable. We thus need to find a way to reduce noise that is generic enough to be deployed in most sensors in use.

1.2 Problem Statement

In sensors, we have constant streams of data flowing in from the environment. Noise in this data flow makes it difficult to have precision in any measurement. In many cases, a reading from an IoT device is likely to be used for further decision making either by humans or by automated means. It is important to have precision.

The aim of this research study was to establish how neural networks and noise filtering algorithms can be used to reduce the noise in data in the most efficient way.

1.3 Objectives

1. To identify the current solutions used to reduce noise in IoT sensors
2. To compare the various noise reduction algorithms used in sensors.
3. To identify how neural networks can be used to incorporate intelligence in sensors.
4. To create and implement a neural network used to reduce noise in IoT sensors.
5. To evaluate the results of using intelligent noise reduction in IoT sensors.

1.4 Significance

Sensors send a constant stream of data of some set parameter from the environment. For most sensors, these are measurements taken per unit time. Usually, the observer has a good idea of what the limits of the readings should be i.e both the lower bound and upper bound of expected reading. Thus, it would be easy to detect outliers in the data as errors.

Noise reduction algorithms can be used but they pose challenges of their own such as accuracy and heavy usage of computing resources. Such resources are very limited in many sensors thus it would not be practical to have them run at all times. We can then utilize neural networks at this point. The network can be taught on what is “noisy” data while using a small portion of the computing resources on the sensor. This way, an energy-efficient filter is utilized.

The study sought to demonstrate how neural networks can be used in IoT devices to filter out the noise in sensors. The study considered various parameters in neural networks. The ANN was tweaked till the optimal combination of parameters could be found for use. The study also compared the success of the ANN to a Kalman filter to establish if it was justified to have an ANN instead of a traditional filter. This information will be useful to the IoT community as well as researchers using IoT devices to collect data.

For other researchers, the study has provided useful insights and reference material on usage of neural networks as a noise filter in IoT devices. The study also exposed areas that will be worth pursuing for further research to expand knowledge.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

IoT devices are basically embedded systems with sensors detecting some environment variable. Any embedded system consists of three parts, sensors, actuators, and controllers. Sensors collect data from the environment and feed it to the controller. The controller then makes a decision based on the input data and sends a command to the actuator to perform some action depending on the data. As such, it is vital that the information collected is correct.

There are various sources of noise in a sensor device as listed by (Lita et al., 2007) namely;

1. Device noise (Noise originating from active and passive devices inside the sensor device)
2. Conducted noise (Noise transmitted in the wires)
3. Radiated noise (Noise from nearby devices)

This noise comes from electronic devices. However, human activity can lead to the introduction of noise in the data. Resetting of devices that are active could actually cause the sensor to report inaccurate data where a data point depends on the previous data point e.g for cumulative values.

Some solutions have been proposed to deal with noise in sensor data and are broadly categorized into two groups:

1. Hardware solutions
2. Software solutions

The two types of solutions are mainly used together and thus complement each other to improve the quality of data. This study details a software-based solution that utilizes machine learning techniques to solve the problem

2.2 Application of Internet of Things

The Internet of Things has many wide applications. Applications are however mainly divided into the following three categories:

1. Smart homes
2. Smart cities
3. Smart industries

Noise could be found in all but is, however, least prevalent in homes due to the controlled environment. Sun Youwei and Su Shaohua (2015) clearly show that where power lines are located close to IoT devices, the data is bound to be noisy. Such heavy power lines are not likely to be found within homes. However, other forms of noise in data can be collected in homes. Some services work better outdoors e.g GPS and thus might report incorrect data if used indoors.

2.3 Electrical Noise

Generally speaking, electrical noise in data is a false reading introduced in electrical devices due to interference by the environment. These electronic devices could be within the actual IoT sensor or without and it will cause some variation in the data collected.

2.4 Categories of Noise in Embedded Systems

Noise in electronic systems is quite common but falls under various categories. According to (Bai et al., 2007), who looked at using RFID tags to collect data, there are various scenarios that can introduce noise in embedded systems.

1. False Negatives

2. False positives
3. Duplicate data.

These categories of noise are quite straightforward. Given RFID tags work by transmitting boolean data, these categories are not sufficient to express the noise that can be found in data streams. As such, another category can be added namely, incorrect data. This is data that is transmitted as a value but such values are not representative of the actual value in the environment.

We looked at the fourth category of data, incorrect data in this study. This is because advanced sensors send an actual value as opposed to either a true or false value.

2.5 Cleaning Noise in IoT data streams

There are various methods used to clean out data streams in IoT data streams. According to (Zhu et al, 2008), they can be classified between local and global solutions.

Local Filtering

Local solutions are solutions based on the sensor. The sensor filters the data locally then transmits the cleaned data. The sensor does not collaborate with any other device to clean out the data. This has a disadvantage in that a faulty or biased sensor can transmit incorrect data. One advantage, however, is that it reduces the data overhead in the network saving bandwidth. The sensor will not transmit faulty data only for it to be discarded later on.

Global Filtering

Global filtering takes advantage of the fact that sensors do not work in isolation but work collaboratively to provide holistic data. Thus, data is transmitted to a centralized node where it is cleaned based on multiple sources of data. This has the advantage of having multiple sources of data to compare against. The disadvantage is that it uses energy and bandwidth to transmit data. Energy in embedded systems is volatile thus any solution needs to consider energy usage in the sensor.

Our study employed a local filtering solution.

2.6 Filtering Methods

2.6.1 Algorithms

1. Kalman Filters

A Kalman filter is an algorithm that uses observable readings over time and uses previous reading to predict missing values or to check the validity of current readings. Sinharay, Pal and Bhowmick, (2011) shows that a Kalman filter works as a typical linear system. Using a Kalman filter, one can give it a set of past values that are correct and it can output a set of valid values with noisy data being filtered out.

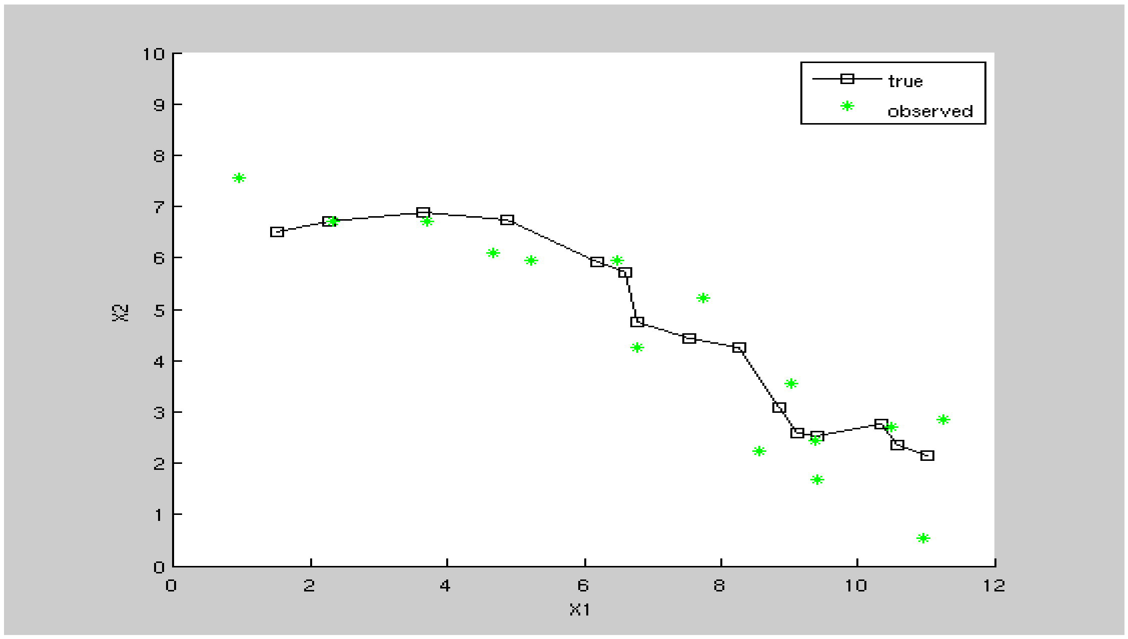


Figure 2.1 Noisy data (Source: http://www0.cs.ucl.ac.uk/)

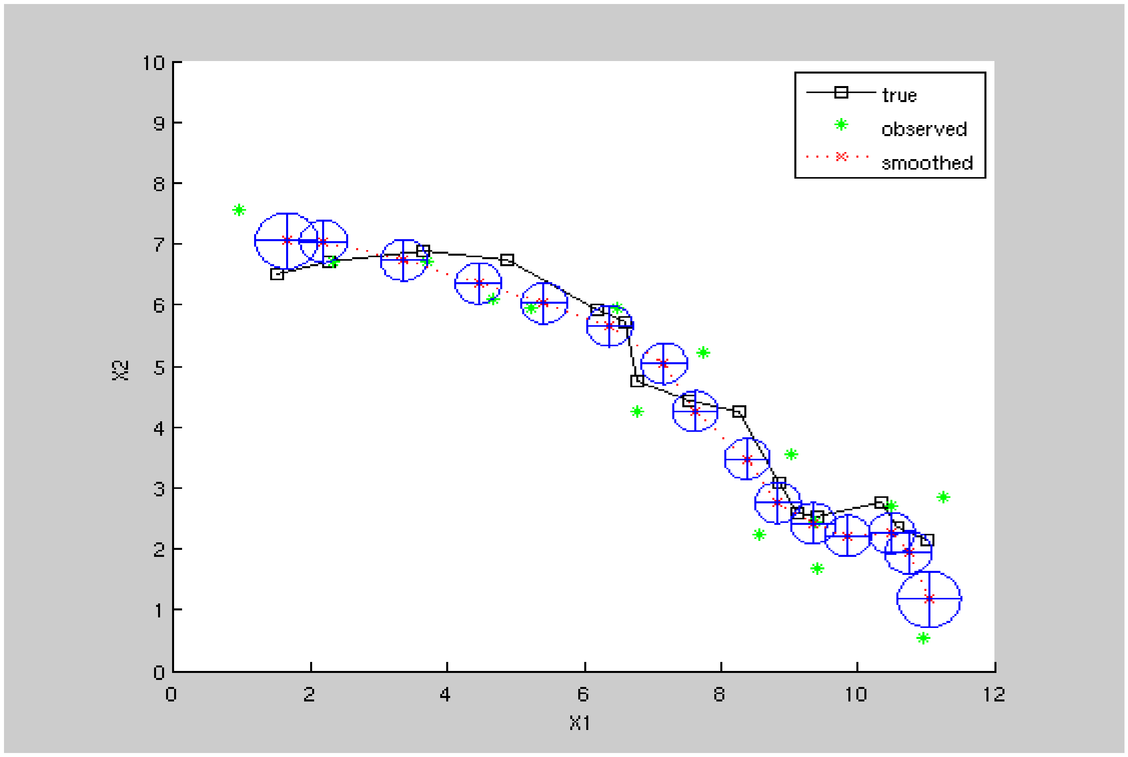


Figure 2.2 smoothened Kalman estimates (source: http://www0.cs.ucl.ac.uk/)

2. Median Filter

A median filter works by passing through the data and replacing a data point with the value of its nearest neighbours. One can set the threshold of how many neighbours to be used in calculating the median.

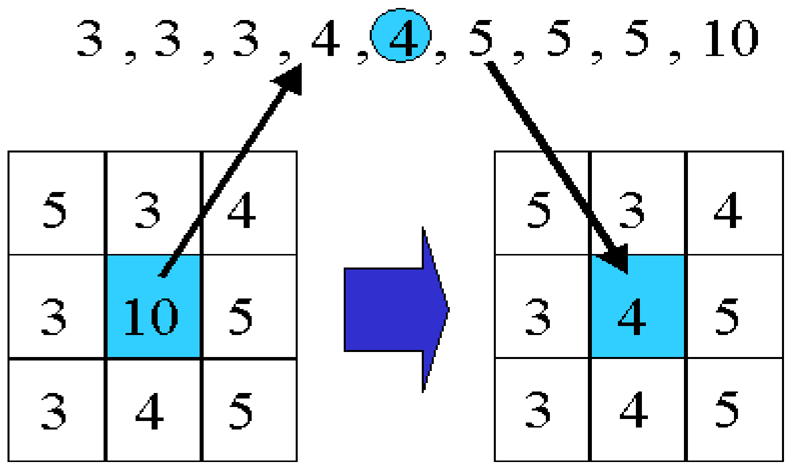


Figure 2.3 Median Filter (Source: mathwork.com)

3. Gaussian Filter

Gaussian filters work to smooth out a signal/stream of data. Common illustrations show a bell curve as the effect of passing a set of data through a Gaussian filter.

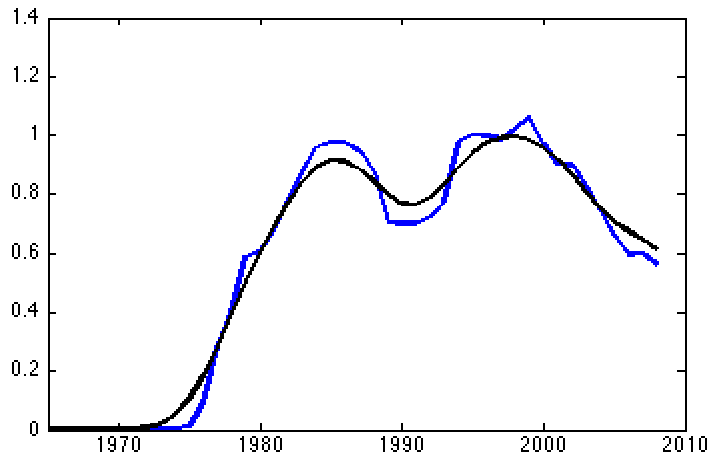


Figure 2.4 Effects of the gaussian filter (Source: mathwork.com)

2.6.2 Use of Neural Networks

Neural networks can also be used to filter out noise in data. Zeng and Martinez (2003) show us that we can use neural networks in two ways to clean out noise in data.

1. Pattern recognition techniques

We can use neural networks to compare data points based on their nearest neighbour so as to detect noise in that data. Nearest neighbour algorithms can be used to detect the expected value of the data and thus clean out erroneous values.

2. Supervised learning

We could use filtering algorithms beforehand to train the neural networks in what should be the correct data. Once trained, the neural networks is deployed with the weights already configured. This way the algorithm is ready for use once deployed in the sensor.

Supervised learning is used to teach the algorithm on the kind of data expected. The Kalman filters was used to provide a comparison to the ANN filter on how well it works as a noise filter. The

Kalman filter was chosen because it uses past data to make decisions which is close to how a neural network works. The neural network was then be deployed in a test environment

2.7 Artificial Neural Networks

Artificial neural networks are conceptual applications that were inspired by the studies conducted on brains and the nervous systems of many animals. It falls under the broader category of artificial intelligence in computer science. Artificial Neural Networks were developed in the 1950s with an aim of imitating the biological brain architecture. Neural networks employ a "divide and conquer" approach to solving big problems (Gershenson, 2003). An Artificial Neural Network basically consists of a set of nodes and the connections between the nodes. Neural networks have been used in a wide array of fields such as Physics, Computer Science, Finance and much more.

2.7.1 How Artificial Neural Networks works

Artificial Neural Networks draw their inspiration from how our brains work and their interconnections. They are inspired by natural neurons found in brains. Natural neurons get a signal via synapses that are located on the dendrites. If the signal is stronger than a certain set threshold, the neuron is activated and thus a signal is emitted. If not, nothing happens.

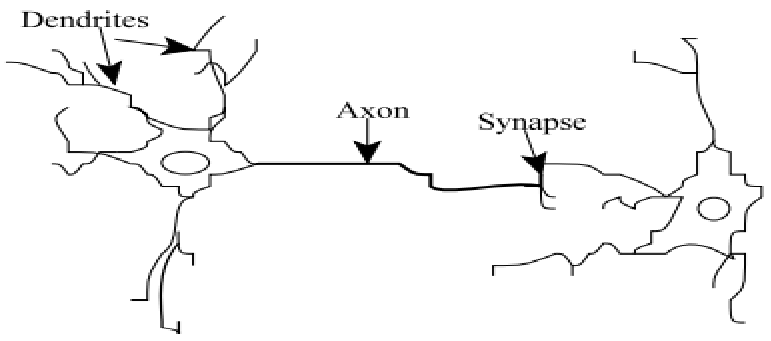


Figure 2.5 Natural neuron (source: Artificial Neural Networks for Beginners)

The nodes are computational units since they receive some input, perform some operation and then produce an output. Sometimes these operations could be simple or complicated. Since they are in a network, information flows from one end to another. The direction of the signal flow could be one way or bidirectional. The various thresholds and operations in the network create a perceivable global behavior of the whole network.

ANNs are designed to find patterns in data over time and use that information to make better decisions. A basic ANN comprises of three layers, input layer, output layer and hidden layers. An ANN has the ability to learn and over time it adjusts its weights and bias to match the desired output.

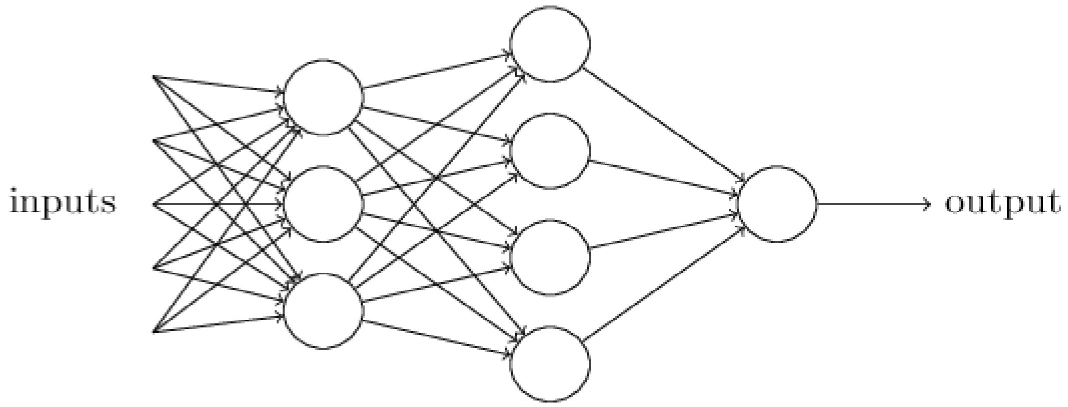


Figure 2.6: Artificial Neural Network (source: http://neuralnetworksanddeeplearning.com/)

A perceptron has several inputs and one output. Each input has weights that are adjusted until the ANN produces the correct results.

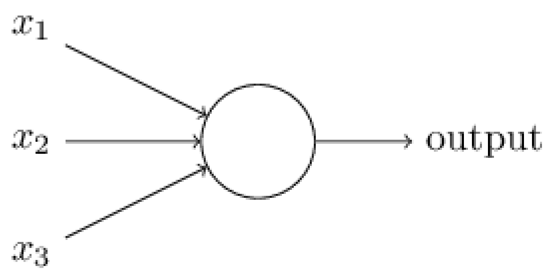


Figure 2.7: Perceptron (source: http://neuralnetworksanddeeplearning.com/)

Each neuron is weighted. The higher the weight, the stronger the input has to be to trigger the threshold. Some weights can also be negative. During training, the weights are adjusted to obtain the desired output based on certain inputs. The processes of finding the optimal weights is learning.

2.7.2 Types of Neural Networks

Maxwell, (2015) shows that some common neural networks are

1. Feedforward networks
2. Feedback/recurrent networks

The classification is based on how data flows through the network.

2.7.2.1 Feedforward Neural Networks

Feedforward networks have an architecture in which no loops exists. The signal flows from the input layer all the way to the output layer without any movement backwards. The weighted connections thus only have their activations fed in a forward direction. The most commonly used feedforward neural networks are multilayered feedforward neural networks. The neurons are arranged into layers comprising of an input layer, an output layer and hidden layer(s). A network could have one or more layers in the hidden layer.

The connections between layers are all unidirectional, moving from the input layer to the output layer. A multilayer feedforward neural network was used this study. We also sought to find the optimal number of neurons in a hidden layer.

2.7.2.2 Recurrent Neural Networks

Recurrent neural networks have loops in their connections. As such, the weighted connections are used to feed previous activations in the network backwards. The data thus flows in both directions.

2.7.3 Training in Neural Networks

Neural networks are used to find the desired output depending on some input provided. The neural network, therefore, has to be trained in how to obtain the correct output given some given input. At the beginning of the training, weight values are randomly assigned and thus the network has no insight into the problem it is being trained on (Maxwell, 2015). With training, the weights are adjusted in a way that the collective network produces the desired output. Training stops when the produced output is at the desired level compared to the expected output.

1. Supervised learning

In supervised learning, both inputs and their respective outputs are provided to the network. The various weights are then adjusted based on the output against the expected output.

2. Unsupervised learning

In unsupervised learning, the correct output is not labelled for the input data. The network finds the correct output by cluster analysis.

3. Reinforcement learning

This one is a combination of both supervised and unsupervised learning. The network penalizes wrong output and rewards correct input.

2.7.3.1 Factors affecting training

1. Epoch size

Epochs are the number of iterations that the training data is passed through the network. In each iteration, the weights are adjusted based on the error calculated.

2. Learning rate

Learning rate is a constant value that is used in a network that employs back propagation to calculate the error and adjust weights. The learning rate affects the speed of learning. With a smaller learning rate, the network takes longer to get to a point where it can stop iterations on the epoch (Maxwell, 2015).

3. Activation function

The activation function is the mathematical formula used to compute the weighted sum of the inputs and produce an output inside a neuron. Activation functions are discussed in detail in section

2.7.4 Training Algorithms used

Training of the neural network is an unavoidable task to get the ANN to work. There are various training algorithms used in ANN and the user has to make decisions on which to use depending on the use. We shall look at the most common training algorithm used in feedforward neural networks and back-propagation algorithm.

2.7.4.1 Back propagation

This algorithm is used in feed-forward ANNs. In a neural network organized in layers, the signal is sent forward. This means that the signal flows from input to the output through the various layers thus it is considered the signal is being sent forward. The signal goes through the various hidden layers which are weighted to provide the desired output.

The backpropagation algorithm uses supervised learning to calculate the error. Examples of the correct results are provided and the results are then compared to the output. The error is calculated and errors are propagated backwards from the output to the input. Initially, the neurons are assigned random weights. As the error is calculated in each cycle, the weights are adjusted so as to reduce the output error. Over time, the network learns the data and gives less erroneous data.

2.7.5 Activation Functions

The activation functions used in neurons can also be referred to as transfer functions. The neurons use the transfer function to produce an output based on the weighted sum of the inputs. The activation functions perform mathematical operations on the input to produce the output. The input layer performs this operation on the training data whereas the hidden and output layers use the output from the interconnection with previous layers as input.

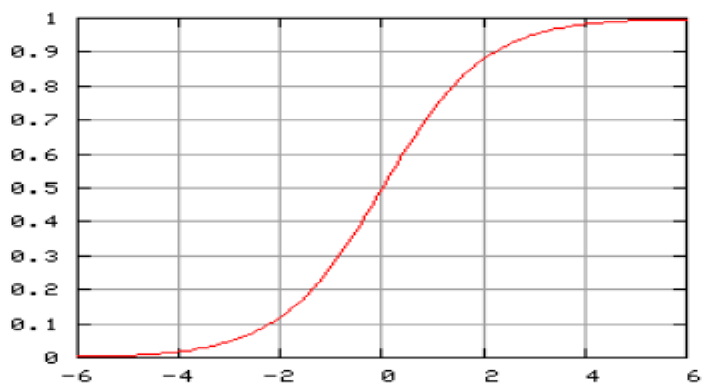
Some of the common activation functions used in neural networks are:

1. Sigmoid (logistic) function

A sigmoid function is a common activation function in neural networks. The formula for a sigmoid function is

```
f(x) = 1/(1+exp(-x))
```

Below is an illustration of the sigmoid function



$$sig(t) = \frac{1}{1+e^{-t}}$$

Figure 2.8 a sigmoid function (source: https://excel.ucf.edu)

2. Tanh function

Kenda et al. (2013) notes that tanh is a rescaled version of the sigmoid function with its output range from -1 to 1 instead of 0 to 1 for the sigmoid function. The formula is

```
f(x) = (2/(1+exp(-2x)))-1
```

$$\tanh(\Sigma) = \frac{e^{\Sigma} - e^{-\Sigma}}{e^{\Sigma} + e^{-\Sigma}}$$

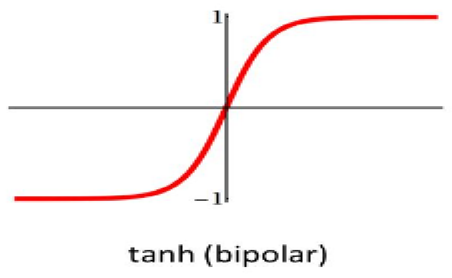


Figure 2.9 Tanh function (source: http://slideplayer.com/slide/5867045/)

3. Linear function

The formula for a linear function is

```
f(x) = x
```

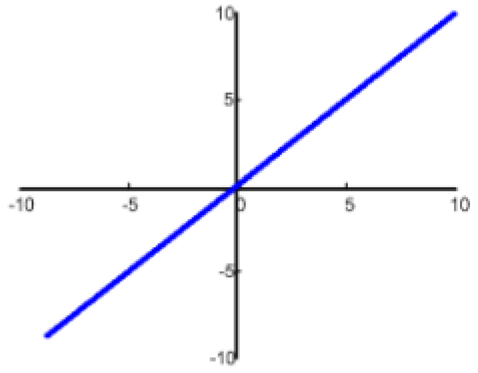


Figure 2.10 A linear function (source: www.intechopen.com)

Maxwell (2015) shows the logistic function is the most common activation function used in neural networks. This is because “it combines nearly linear behavior, curvilinear behavior and nearly constant behavior, depending on the value of the input.” Our study has looked at the effects of various activation functions on the results.

2.8 Uses of Neural Networks in IoT

Neural networks are used in sensors to make them intelligent improving their efficiency. Razafimandimby et al. (2016) demonstrate that using neural networks, robots can actively cooperate and exchange data which makes them more efficient over time. Such robots collect data and learn on how to better cooperate and be more efficient.

Wu et al. (2013) proposes a neural network solution that will be used to detect situations in coal mines. Due to the intelligence of neural networks, the solution is able to identify common situations in coal mines early as well as identify anomalies.

2.9 Kalman Filter

The Kalman filter is used to provide a solution to filtering on data accurately (Haykin, 2001). It is a recursive solution that makes use of past predictions to filter out current data. Kalman filter works well with linear time series data (Yonglong and Dongyan, 2015).

However, the Kalman filter is often used with other filters to improve accuracy. Kalman filters are currently widely used together with recurrent neural networks to solve problems. In (Alessandri et al., 2003) and (Sum et al., 1999), it is used to train neural networks. In (Sum et al., 1999) it is also used to prune a feed forward neural network.

The Kalman filter is essentially useful to predict future data from past data. The more data that comes in the better decision the filter can make. The difference with other filters is that the Kalman filter does not wait for all the data to arrive rather it makes predictions as more data arrives. Laaraiedh (2016) provides the formula

The formula tries to estimate the state $x \in \Re^n$ of a discrete-time controlled process that is governed by the linear stochastic difference equation

$x_k = Ax_{k-1} + Bu_k + w_{k-1}$

with a measurement m $y \in \Re^m$ that is

$y_k = Hx_k + v_k$

W_k and v_k are random variables that represent process and measurement noise.

2.10 Use of Sensors

IoT sensors are mainly passive. They collect data from a specific environment variable and send that to some central location for a decision to be taken. Razafimandimby et al. (2016) shows that in an environment where robots need to work together to achieve a common task, they can use IoT sensors to communicate and share information about their location to ensure they are always in communication range. This thus ensures global connectivity of the Multi-Robot System.

2.11 Noise control in sensors

When dealing with any electric devices, errors are inevitable. IoT devices are however quite susceptible to electronic interference. Several techniques to mitigate this noise have been proposed. Galambos and Sujbert (2015) proposes an Active Noise Control for IoT. A variant of Least Mean Squares algorithm is used to actively detect and remove noise from a microphone before transporting the result via an ethernet cable. Sun Youwei and Su Shaohua (2015) on the other hand focuses on noise from power lines on IoT sensors. Robust Independent component analysis algorithm was used to effectively improve the signal-to-noise ratio. Zhang et al. (2011) uses a wavelet function to reduce noise in an intelligent building. A wireless sensor network is setup in the building to collect data on temperature. A threshold is set and the data is passed through the wavelet function and the data is reconstructed removing the noise. Das and Panda (2004) looks to deal with noise pollution where speakers are in use. The study uses a similar technique to (Galambos and Sujbert, 2015) but opts for an artificial neural network to do the filtering. Raeisy and Golbahar Haghighi (2012) also uses a neural network to filter the data for active noise control.

2.11.1 Error Rates in Sensors

Data in sensors is bound to be noisy because of various factors such as nearby electronic devices and faulty equipment. Errors are caused by various factors as noted by (Elnahrawy and Nath, 2003). Some of the factors include:

1. Software errors
2. Unreliable communication channel
3. Poorly calibrated equipment

These factors contribute to high error rates in sensor data especially when you have inexpensive equipment. How to deal with the errors has been handled in other parts of literature. McHenry et al. (2015) shows that while near high voltage lines, the collected noise via IoT is likely to be 10% higher than under normal circumstances. The study however also notes that a complete study into RF noise is lacking since over 30 years ago and thus it is difficult to give informative figures on the error rates in IoT.

2.12 Conceptual Framework

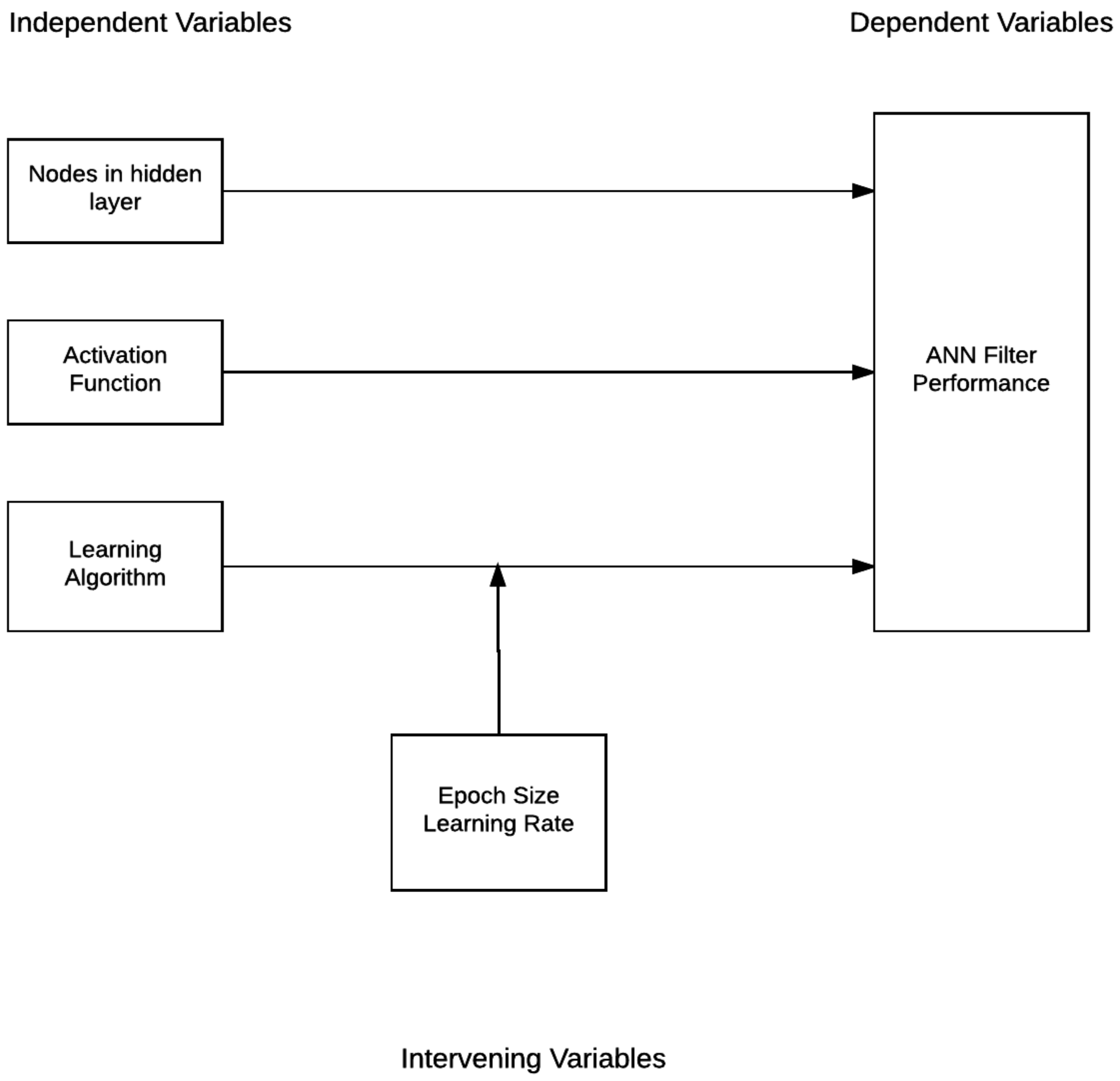


Figure 2.11 Conceptual Framework

According to the diagram, ANN filtering performance is affected by the ANN architecture which is the number of nodes in the hidden layer and the activation function used. The intervening variables are in this case the epoch size and the learning rate parameters which have been observed in literature to have major contributions to neural network performance.

CHAPTER 3: RESEARCH DESIGN AND METHODOLOGY

3.1 Research Design

The methodology employed for this study was quantitative using applied research method. In the study, a specific intelligent tool was created based on a specific model and tested for a particular set of data. An alternative tool was also developed in comparison to the intelligent tool. Chapter 2 showed there is need for a tool to filter out noise in data.

An Artificial Intelligence model was designed based on an ANN algorithm. The algorithm was then used to create a working prototype to test the model. The ANN that was the output of this process was a multi-layer perceptron using backpropagation as the training algorithm. The ANN was trained using supervised learning.

The Kalman filter was also designed as a tool for comparison with the ANN. It was designed based on the mathematical formula:

$$x_k = Ax_{k-1} + Bu_k + w_{k-1}$$

Both algorithms were developed using the python programming language. This required having the python environment working on a computer while testing. Other requirements for the application were documented in a requirements file to assist in quick setup of the environment.

The noise filter could be used generically across any domain with data that is noisy. The focus of this study however was IoT devices and thus the filter was tested against simulated noisy temperature data.

3.2 Research Data

3.2.1 Source of Data

The data used to evaluate the ANN and Kalman filter tool was simulated to reflect what a faulty sensor would send in. A python script was created to generate noisy data that follows an upward trend. This enabled a direct simulation of the data streams coming in from sensor devices deployed in some environment. The data was fed into the ANN tool for both training and testing. The holdout method was chosen in which some data was used for training the ANN and a smaller portion of the data was used for testing. The smaller portion would then prove the accuracy of the filtering tool be it the ANN or the Kalman filter. In the study, there was one input and one output on the ANN tool. The input was the noisy data and the output would be compared to the expected value where the error would be corrected. Using back-propagation, the ANN tool would then be trained using the training data. Once the training phase was done, the testing data was fed in as input and the output compared to the true value.

3.2.2 Volume of Data

In the study, 100 data points were used to simulate noisy data points from a sensor. This volume would approximately be a day's data reading. This corresponds to a reading being recorded every quarter hour. The approach was considered to be sufficient for training and testing the ANN tool as well as for use by the Kalman filter tool.

3.3 Designing the Proposed Model

The proposed model had one input neuron and one output neuron. The model was designed to have a variable number of neurons in the hidden layer and that was tested. The number of neurons in the hidden layers was varied and an optimal configuration was found. The same case was applied to the activation function. Two activation functions were considered, sigmoid/logistic and the tanh activation functions. The proposed model, was then developed as discussed in chapter 4.

3.4 Application Development

In this study, the application was implemented using the Rapid Application Development (RAD) software development process. This involved prototyping and improving the prototype. This method worked well in this study since it was a research project with a prototype and the various steps of development were not well defined. The overall result of using RAD was very high-quality software developed very quickly which was the point of this study.

3.4.1 Justification of Using RAD

This methodology offers a viable option since it enables software to be developed quickly with the requirements not well defined beforehand. In this study, the requirements could have easily changed during development.

The time given to complete the project is quite constraining also. The study period was 6-8 months which was not enough to create a conclusive product using traditional methodology such as the waterfall model. CASE tools have also been developed which were a huge help in creating the solution. With these tools, we were able to design and implement the application faster and more feasibly making RAD a very viable option.

Finally, adopting this methodology, we were at liberty to improve the application with ease. This demanded developing the application using an object-oriented approach. The various functionalities were modular.

3.4.2 RAD Lifecycle

Below is a brief overview of the RAD process, which consists of four lifecycle stages: Requirements Planning, User Design, Construction, and Implementation.

3.4.2.1 Requirements Planning Phase

The requirements planning stage consisted of developing a high-level list of initial requirements as well as setting the research scope. This stage involved reviewing of current solutions and talking to the users of such systems.

3.4.2.2 User Design Phase

The various requirements were collected and modelled. A prototype of the user interface was created to communicate the idea behind the final solution. This stage was a continuous interactive process that allowed the understanding of the proposed solution, modifying, and eventual approval of a working model of the system.

3.4.2.3 Construction Phase

Here, the application was developed through iterative prototyping. Specific modules were created in isolation till they were operational. Input from various stakeholders was also taken into account to accommodate the various requirements. Once the specific modules were created, they were loosely coupled and designed to communicate via API calls.

3.4.2.4 Implementation Phase

As soon as the application was stable and is deemed to be ready for use, it was deployed. Here all functionality that were turned on for debugging were turned off. All resources needed for deployment were also be provisioned and configured.

3.4.3 Application Testing

Once the application was developed and deployed there was the need to verify that it was working in reducing noise in data. The solution was run through a series of tests. In order to validate the application and guarantee its correctness, the specifications and properties that we were trying to prove were verified. Three verification goals were focused on:

1. Each component of the application must work correctly when considered in isolation. As such, unit tests were written for each module and updated from time to time once the code changed.
2. After these individual components were proven to work, they were integrated then run through rigorous black box testing to ensure they can correctly work together. Some more tests were written as part of the application code that tested the end to end functionality.

3. The application was then deployed in a computing device where it can be tested. Noisy data was generated to simulate a sensor in the field.

3.5 Project Design

This research employed a modular approach to developing software where modules were developed in an incremental manner. This was to adhere to the rules of RAD. Each module was developed then tested and if it was working, the next module was created and added to the existing one. This cycle continued till the project was completed.

The underlying architecture used is client-server architecture. This is because the sensors are perceived as clients transmitting data to a centralized server. Data processing also happened at the simulated IoT device. The presentation, views, and the data management were logically separate processes on the IoT device.

The application had a user interface, middleware for handling sensor requests and the database. Multi-tier application architecture provided a model that enables the creation of a flexible and reusable application by breaking it up into several manageable application tiers. This enabled each tier to be handled separately which was very convenient.

Typically, the user interface runs on a web browser. The UI was accessible to any device within the network thus it was deployed as a typical web application. Application logic consisted of one or more separate modules running on the server controlled by a web server. The data was managed using an RDBMS on a database server.

3.6 Application Architecture

The Artificial Neural network was designed and integrated as part of the main application. The conceptual flow of how it would work was designed. Figure 3.1 is a diagram of its architecture.

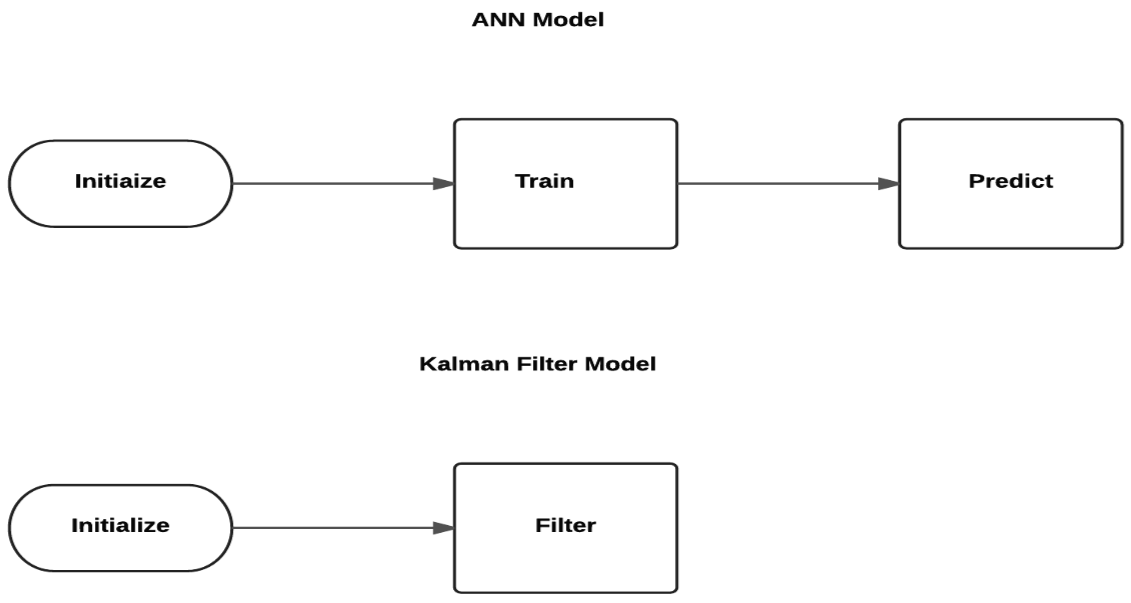


Figure 3.1 ANN and Kalman Filter Model

The neural network was broken down into three parts, initialization of the ANN, the training and finally the prediction part. This conceptual diagram was helpful and the application was designed to have the three parts separate. The Kalman filter was simpler to implement and thus only had two parts. Once the initialization happened, the prediction worked by running the data through the filter any desired number of times.

The application was developed in python programming language and the code is provided in the appendix section. The applications could be accessed via an API that would pass the desired parameters to filter and return the results as a JSON string. A simple user interface was developed to display the results of call.

CHAPTER 4: RESULTS ANALYSIS AND PRESENTATION

4.1 Introduction

This chapter presents findings of the study where various factors that influence the effectiveness of the neural network were examined. The study also compared the results of the neural network and the Kalman filter.

At this stage, the tools were designed and run with the variables tweaked at each stage. After this, the results were observed. These tests were run on both the Kalman filter and the ANN. The following metrics were to be observed and recorded

1. Accuracy
2. Speed
3. Complexity

4.2 Artificial Neural Network

Once the ANN was designed and deployed in a testing environment, the following variables were modified and the effect was observed.

1. Training data
2. Epochs
3. Hidden Layers
4. Activation function

4.2.1 Varying epoch sizes

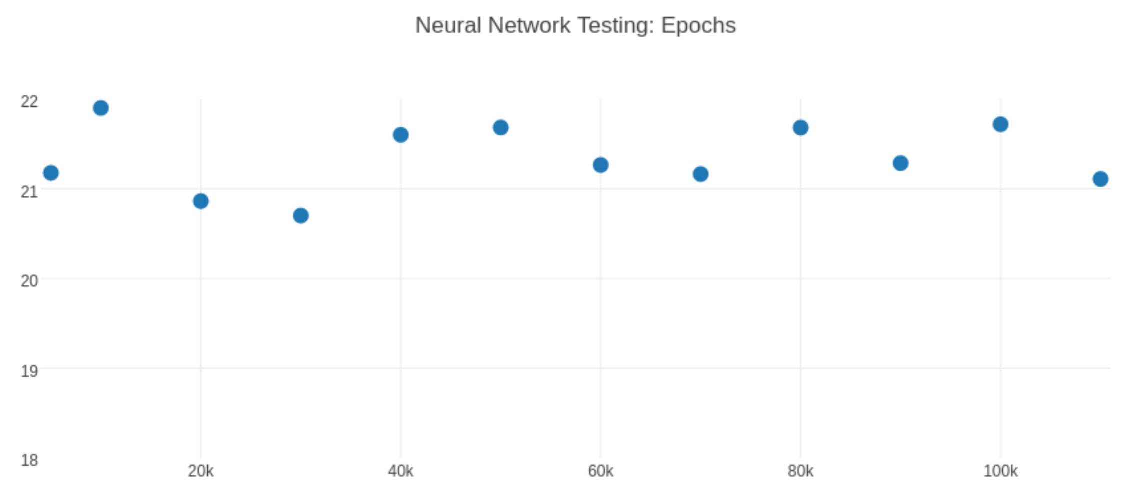


Figure 4.1 ANN testing: epochs

In varying the size of the epoch, it was observed that the network get closer to the correct prediction at 40,000 epochs. It was thus noted, to obtain reliable results, the training data had to be iterated over at least 40,000 times. After 40,000 iterations, the results remained largely close to 22 with the largest variance being 0.9 at 110,000 epochs. For our study, however, 50,000 was used because it had a smaller error compared to 40,000 epochs. This was done with three hidden layers and the tanh activation function. It was also observed that at 10,000 epochs, a prediction close to the expected result was found but based on results from close range epochs, the result was ignored as a "good guess."

Epochs	Hidden Layer size	Activation Function	**Output**	**Error/ Variance**	Training /Expected value
5000	3	tanh	**21.18**	**0.82**	22

10000	3	tanh	**21.90**	**0.10**	22
20000	3	tanh	**20.86**	**1.14**	22
30000	3	tanh	**20.70**	**1.30**	22
40000	3	tanh	**21.60**	**0.40**	22
50000	3	tanh	**21.68**	**0.32**	22
60000	3	tanh	**21.27**	**0.73**	22
70000	3	tanh	**21.16**	**0.84**	22
80000	3	tanh	**21.68**	**0.32**	22
90000	3	tanh	**21.29**	**0.71**	22
100000	3	tanh	**21.72**	**0.28**	22
110000	3	tanh	**21.11**	**0.89**	22

Table 1: ANN using tanh function and 3 neurons in the hidden layer with varying epochs

As noted earlier in literature, the sigmoid function is similar to the tanh activation function with the exception of having a smaller range(0 to 1) compared to the tanh (-1 to 1). The tests were repeated using the sigmoid function and these were the results

Epochs	Layers	Activation Function	**Output**	**Error/ Variance**	Training /Expected value
5000	3	sigmoid	**19.90**	**2.10**	22
10000	3	sigmoid	**19.80**	**2.20**	22
20000	3	sigmoid	**20.40**	**1.60**	22
30000	3	sigmoid	**20.38**	**1.62**	22
40000	3	sigmoid	**19.75**	**2.25**	22
50000	3	sigmoid	**19.88**	**2.12**	22
60000	3	sigmoid	**20.25**	**1.75**	22
70000	3	sigmoid	**20.31**	**1.69**	22

80000	3	sigmoid	**20.70**	**1.30**	22
90000	3	sigmoid	**20.00**	**2.00**	22
100000	3	sigmoid	**20.99**	**1.01**	22
110000	3	sigmoid	**20.91**	**1.09**	22

Table 2: ANN using sigmoid function and 3 neurons in the hidden layer with varying epochs

The results indicate that the sigmoid function is more erroneous compared to the tanh activation function. The smallest error using a tanh function was 0.10 compared to 1.01 using a sigmoid function. While using the sigmoid function, the errors seem to be higher compared to the tanh function with the same number of epochs. As such, the tanh function was used as the activation function for this study.

4.2.2 Neurons in Hidden layer testing

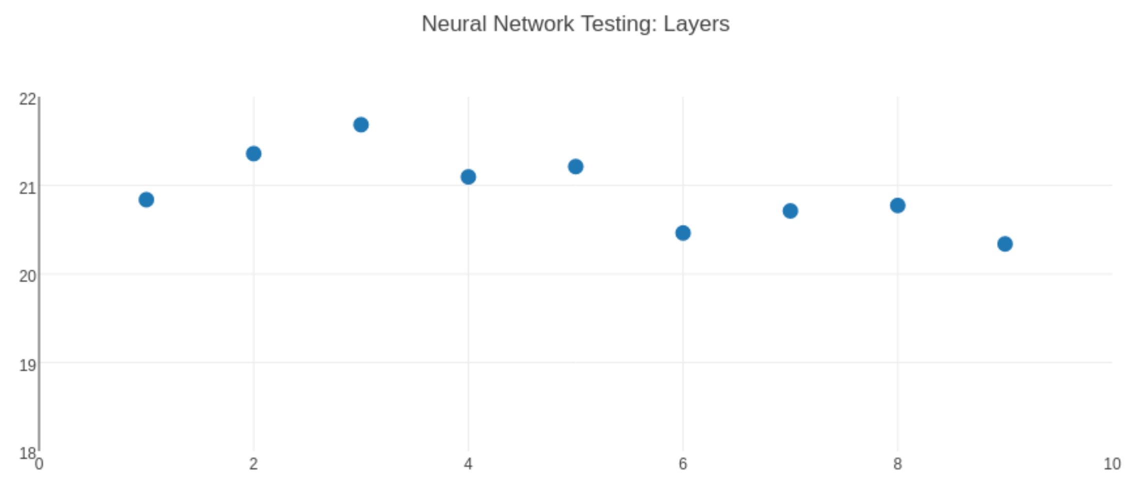


Figure 4.2 ANN testing: neurons in hidden layer

The number of neurons in the hidden layer was varied and the results observed. 2 neurons were observed to be sufficient in the hidden layer. The best results were however observed to be at 3 neurons in the hidden layer. Additional neurons added complexity without improving results greatly on the neural network.

Epochs	**Layers**	Activation Function	**Output**	**Error/ Variance**	Training /Expected value
50000	**1**	tanh	**20.84**	**1.16**	22
50000	**2**	tanh	**21.36**	**0.64**	22
50000	**3**	tanh	**21.68**	**0.32**	22
50000	**4**	tanh	**21.09**	**0.91**	22
50000	**5**	tanh	**21.21**	**0.79**	22
50000	**6**	tanh	**20.46**	**1.54**	22
50000	**7**	tanh	**20.71**	**1.29**	22
50000	**8**	tanh	**20.77**	**1.23**	22
50000	**9**	tanh	**20.34**	**1.66**	22

Table 3: ANN using sigmoid function and 50000 epochs with varying hidden layer size

4.3 Kalman Filter

The Kalman was designed and deployed in a testing environment, the iterations were varied and the effect was observed.

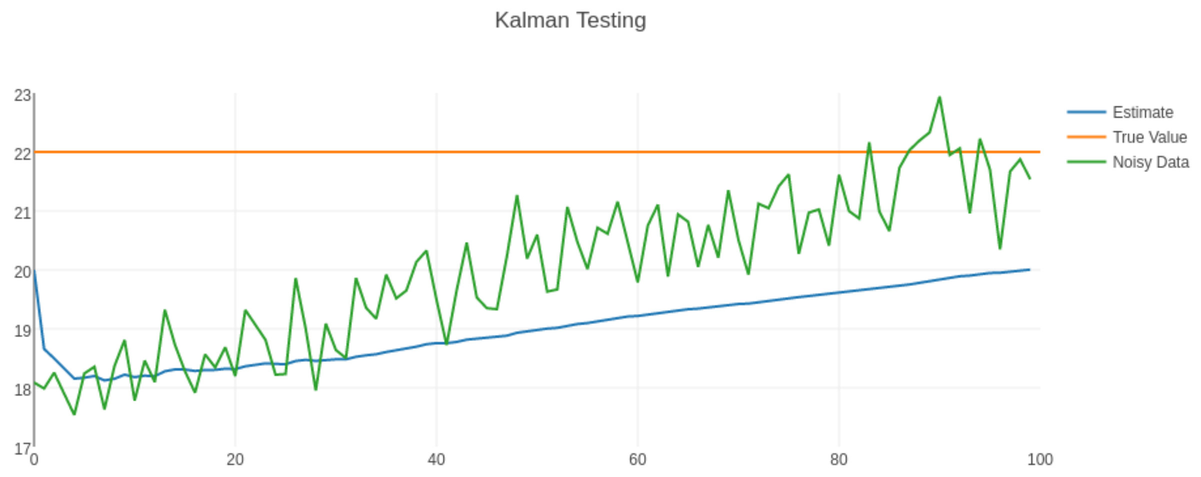


Figure 4.3 Kalman testing

The Kalman filter was used to compare how effective the ANN network was. The Kalman filter was observed to converge faster to the estimated answer but made huge errors compared to the ANN over time. Below are the results over time.

Iterations	Initial guess	Prediction	Expected result	Error	Time (seconds)
50	20	18.99	20.5	1.51	0
100	20	20	22	2.0	0

Table 4: Kalman filter predictions at varying iterations

From the data above, it can be observed that the Kalman filter is quick to make a guess. This is based on the amount of time in seconds as shown in the table plus the diagram above. But the estimation gets worse as the iterations grow. In fact, the error was less after 50 iterations compared to 100. As such, despite the Kalman filter having quick convergence time, the error rate is higher compared to the ANN. The Kalman filter however is able to observe the trend and keep with the pattern.

4.4 Kalman Vs. ANN results

The ANN is proven to produce better results than a Kalman filter from the results above. The ANN is bound to make more erratic errors with few iterations but the variance from the estimated result is still lower compared to the prediction from the Kalman filter. One major advantage of the ANN is that it produces better results over time whereas the Kalman filter is not observed to produce better results over time. This points to having a neural network as a more reliable filter in sensors.

In terms of complexity, the Kalman filter is easier to implement and get working. The ease, however, comes at a cost in accuracy. Modern devices have sufficient resources to run complex applications thus complexity is not a major factor to consider in IoT sensors. Accuracy, however, is key and should be considered as an important factor in determining the tool to use. In this case, neural networks come out winning. The code for each tool can be seen in the appendix section.

4.5 Performance Metrics

As noted in literature, sensors have very limited resources compared to traditional computing devices. They have limited memory, disc space and processing capability. As such, it was necessary to also observe how the two solutions performed. During testing, time taken to perform operations was collected and stored. Each took less than a second. However, the Kalman filter was found to be faster than the ANN at the same number of iterations. At 100 iterations, the Kalman filter took 0.0432 sec while the ANN took 0.6832 sec. That implies that the Kalman filter is 15 times faster than the ANN. This was due to the loops in the ANN increasing complexity.

4.5.1 Speed

Each filter was fast under 50,000 iterations and produced results in less than a second. However, as the number of iterations grew, the neural network was shown to be slower under the same environment. Based on the timing tests, the Kalman filter was found to be 15 times faster than the ANN at the same number of iterations.

4.5.2 Accuracy

From the results shown above, it was proven beyond doubt that the ANN was by far more accurate than the Kalman filter. This was especially the case after more data was passed to the filters. The Kalman filter does slowly converge towards the correct answer over time but results are still highly inaccurate. The ANN, however, does improve its results over time but does reach a peak at some point where more iterations on training to not produce better results. As such, for accuracy, the ANN is by far the better alternative.

The two filters thus have strengths and weaknesses in various areas. The ANN, however, seems to emerge as a better filter since even at small iterations, it does produce better data compared to the Kalman filter.

CHAPTER 5: DISCUSSIONS, CONCLUSIONS AND RECOMMENDATIONS

5.1 Introduction

The purpose of this study was to investigate how to use neural networks to filter out noise in IoT sensors. Various variables on both the neural network and the Kalman filter were examined and optimal configurations were identified. The two filters were then compared to each other.

5.2 Summary of Findings

The study findings were as follows:

5.2.1 The tanh activation function works better than the sigmoid function.

In the study, the tanh activation function was found to consistently perform better than the sigmoid function when it comes to predicting. When the sigmoid function was used, the data was more erroneous. As such, the activation function used in our study was the tanh activation function

5.2.2 The ANN produced optimal results at 50,000 epochs.

In the study, it was found that the ANN works best at 50,000 epochs. It did occur that best results were found at lower epochs but based on results of almost similar iterations, it was deemed to be inconclusive and could be the ANN got lucky. The best results that were deemed stable were found at 50,000 epochs and the prediction did not vary much after that.

5.2.3 Three neurons in the hidden layer produced best results.

The best results in the ANN were found when using three neurons in the hidden layer. Three neurons produced the lowest error compared to the other hidden layer sizes. As such, three neurons in the hidden layer were adopted as the hidden layer size for this study.

5.2.4 The Kalman filter is a good alternative to ANN but not as accurate.

It was identified that the Kalman filter is a good alternative to the ANN filter. However, the filter had larger errors in prediction compared to the ANN based on the data at hand. The Kalman filter however is able to make predictions faster than the ANN. The Kalman filter was also able to find the pattern in the data despite the error making it a good alternative to the ANN filter.

5.3 Discussions

The first objective of this study was to identify the current solutions used to reduce noise in IoT sensors. It was determined that intelligent approaches are used to filter out noise in data as well as other filter solutions such as the Kalman filter and the median filter. The median filter was discussed in the literature while a Kalman filter and an ANN were tested and the performance was compared. Both can perform well as filters in noisy data.

The second objective of this study was to compare the various noise reduction algorithms used in sensors. The Kalman filter algorithm was looked into and the median filter. The Kalman filter was implemented and tested. Results of the Kalman filter were collected and recorded. The Kalman filter is fast, compared to a neural network based filter and can find the noise pattern. However, the error rate was high compared to the ANN.

The third objective was to identify how neural networks can be used to incorporate intelligence in sensors. This was discussed in literature and it was shown that other studies have used ANNs to filter out noise in data. An ANN was developed in this study and the results were promising. With the right configurations of its parameters, the ANN was found to work well as a noise filter in IoT sensors.

The fourth objective was to create and implement a neural network used to reduce noise in IoT sensors. The neural network was implemented and tested. The application can be found in the appendix section. The findings show that the developed ANN can perform well as a noise filter compared to other filters.

The last objective was to evaluate the results of using intelligent noise reduction in IoT sensors. The results were positive based on our findings. The ANN was found to be less erroneous compared to the Kalman filter despite being slower and more complex.

5.4 Conclusion

In the study, the potential of using an ANN as a filter in cheap IoT devices was explored. This confirmed the findings of (Raeisy and Golbahar Haghighi, 2012). An alternative filter, the Kalman filter was developed and used to compare its potential as a filter compared to the ANN. The study relied on simulated data that was used to provide input to both filters with the expected output also generated.

The neural network was found to be optimal with a tanh activation function and 3 neurons in the hidden layer. Back-propagation training method was used for the ANN and results were good as shown in chapter 4. Finding the correct number of neurons in the hidden layer was a trial and error process and after some tests, 3 was found to be an optimal number.

A basic Kalman filter was also developed to test its performance as a reliable filter for noisy data. Though quite fast in making a prediction, the filter was found to be more erroneous compared to the ANN and thus was found to be useful but not ideal in predicting correct values from noisy data. As computing resources keep growing cheaper, speed can be improved over time by obtaining sensors with better resources. Performance, however, is mainly handled on the software end in which the ANN comes out ahead. The filter was a simple solution to create too compared to the ANN.

It was thus observed that using an ANN filter is better compared to a Kalman filter. Results for each filter were compared with the ANN performing better in terms of accuracy of the prediction. The ANN was however found to be slower as the number of iterations grew where the Kalman filter was quite fast even with large iterations. The main undoing however for the Kalman filter was that even as the number of iterations grew, the Kalman filter was not able to converge to an accurate result in contrast to an ANN. It was thus observed that a neural network solution would provide a better alternative.

The study also looked at the various factors that influence the performance of the ANN. Here, the activation function used was tested, the number of neurons in the hidden layer as well as the number of epochs when training the ANN. Two activation functions, the sigmoid function and the tanh function were tested. Both functions performed well but the tanh activation function was found to produce better results. The error on the predicted data compared to the expected result was found to be smaller using the tanh function after various tests where the number of epochs was varied. The epochs when training the network were also tested and the network would provide best results at 50,000 iterations after which more epochs would not improve the prediction. The number of neurons in the hidden layer was also tested and 3 was found to be the optimal value. The result did not improve as the number of neurons grew.

5.5 Recommendations

- Collect data on error rates in sensors. It was noted during the research project that there is little data regarding error rates in IoT sensors. McHenry et al. (2015) shows it is a major problem but it is difficult to qualify that assertion with data. As such, research into error rates in IoT devices is a highly recommended research area.
- The study recommends exploring other training methods such as Levenberg-Marquardt Optimization Algorithm to improve ANN performance. The data collected was tested using the most common training method, back-propagation. This was because of the amount of time and scope of the research. However, the study would benefit greatly if other training methods were used and compared to back-propagation method used to train the data in the current study.
- The study recommends trying other activation functions other than the sigmoid and tanh activation function and compare performance for reasons similar to those listed above for the training function
- The study would recommend combining the Kalman filter and ANN to improve the results of the IoT filter. This would have an integrated application that ensures that both filters are used together with the Kalman filter complementing the ANN.
- The study simulated data in a lab setting for use in testing. More research can be conducted by collecting data from IoT sensor that have been deployed in the field with ANN filter to test performance and also collect data for training other ANN filters.

- The study focused on constant linear values as test data. Research can be conducted on the performance of both filters when having non-linear data that is noisy.
- More research can be conducted into how the ANN filter performs against other filters except the Kalman filter
- Finally, research can be conducted into the performance of the ANN filter in other fields other than IoT sensors. This can include filtering noise in data collected via conventional means such as questionnaires.

References

A. Nielsen, M. (2015). Michael A. Nielsen, *Neural Networks and Deep Learning*. 1st ed. Determination Press. - Not user

Alessandri, A., Cirimele, G., Cuneo, M., Pagnan, S. and Sanguineti, M., (2003) *EKF learning for feedforward neural networks, European Control Conference (ECC)*, Cambridge, UK, 2003, pp. 1990-1995.

Bai, Y., Wang, F., Liu, P., Zaniolo, C. and Liu, S. (2007). *RFID Data Processing with a Data Stream Query Language*. 2007 IEEE 23rd International Conference on Data Engineering.

Das, D. and Panda, G. (2004). *Active Mitigation of Nonlinear Noise Processes Using a Novel Filtered-s LMS Algorithm*. IEEE Transactions on Speech and Audio Processing, 12(3), pp.313-322.

Elnahrawy, E. and Nath, B., (2003). *Cleaning and querying noisy sensors*. In Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications (pp. 78-87). ACM.

Frenzel, L. (2016). *Will Noise and Interference Throttle the Internet of Things?*. [online] Electronicdesign.com. Available at: http://electronicdesign.com/blog/will-noise-and-interference-throttle-internet-things. [Accessed 6 Oct. 2016].

Galambos, R. and Sujbert, L. (2015). *Active noise control in the concept of IoT*. Proceedings of the 2015 16th International Carpathian Control Conference (ICCC).

Gershenson, C. (2003). *Artificial Neural Networks for Beginners.* [online] Arxiv.org. Available at: https://arxiv.org/abs/cs/0308031 [Accessed 14 Jul. 2016].

Haykin, S. (2001). *Kalman filtering and neural networks*. 1st ed. New York: Wiley.

Kenda, K., Škrbec, J. and Škrjanc, M., (2013). *Usage of the Kalman Filter for Data Cleaning of Sensor Data*. proceedings of IS (Information Society).

Laaraiedh, M. (2016). *Implementation of Kalman Filter with Python Language.* [online] Arxiv.org. Available at: http://arxiv.org/abs/1204.0375v1. [Accessed 16 Jun. 2016].

Lita, I., Visan, D., Oprea, S. and Cioc, B. (2007). *Hardware Design for Noise Reduction in Data Acquisition Modules*. 2007 30th International Spring Seminar on Electronics Technology (ISSE).

Maxwell, C. Odira. (2015) *Prediction of River Discharge Using Neural Networks*, University of Nairobi eRepository

McHenry, M. A., Robertson, D., and Matheson, R. J. (2015), *Electronic noise is drowning out the Internet of things*, IEEE Spectrum: Technology, Engineering, and Science News, Available at: http://spectrum.ieee.org/telecom/wireless/electronic-noise-is-drowning-out-the-internet-of-things. [Accessed: 3 Jun. 2016].

Raeisy, B. and Golbahar Haghighi, S. (2012). *Active Noise Controller with reinforcement learning*. The 16th CSI International Symposium on Artificial Intelligence and Signal Processing (AISP 2012).

Razafimandimby, C., Loscri, V. and Vegni, A. (2016). *A Neural Network and IoT Based Scheme for Performance Assessment in Internet of Robotic Things*. 2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI).

Sinharay, A., Pal, A. and Bhowmick, B. (2011). *A Kalman Filter Based Approach to De-noise the Stereo Vision Based Pedestrian Position Estimation*. 2011 UkSim 13th International Conference on Computer Modelling and Simulation.

Sum, J., Leung, C., Young, G. and Kan, W. (1999). *On the Kalman filtering method in neural network training and pruning*. IEEE Transactions on Neural Networks, 10(1), pp.161-166.

Sun Youwei, and Su Shaohua, (2015). *Research on noise reduction of internet of things based on power line using independent component analysis*. 2015 International Conference on Information and Communications Technologies (ICT 2015).

Wu, X., Wu, J., Cheng, B. and Chen, J. (2013). *Neural Network Based Situation Detection and Service Provision in the Environment of IoT*. 2013 IEEE 78th Vehicular Technology Conference (VTC Fall).

Yonglong, Y. and Dongyan, C. (2015). *Globally optimal Kalman filtering with correlated noises, random one-step sensor delay and multiple packet dropouts*. 2015 34th Chinese Control Conference (CCC).

Zeng, X. and Martinez, T. (2003). *A noise filtering method using neural networks*. IEEE International Workshop on Soft Computing Techniques in Instrumentation, Measurement and Related Applications, 2003. SCIMA 2003..

Zhang, Z., Fang, Q., Cheng, H. and Cheng, L. (2011). *Wavelet transform based noise reduction method for temperature data sequence in intelligent building*. Proceedings of 2011 International Conference on Computer Science and Network Technology.

Zhu, X., Zhang, P., Wu, X., He, D., Zhang, C. and Shi, Y. (2008). *Cleansing Noisy Data Streams*. 2008 Eighth IEEE International Conference on Data Mining.

APPENDICES A:

ANN filter python code

```
from __future__ import division

import numpy as np

def tanh(x):

    return np.tanh(x)

def tanh_deriv(x):

    return 1.0 - np.tanh(x)**2

def logistic(x):

    return 1/(1 + np.exp(-x))

def logistic_derivative(x):

    return logistic(x)*(1-logistic(x))

def _scale_to_binary(e, minV, maxV):

    result = ((e-minV)/(maxV-minV))*(1-0)+0

    return result

def rescale_from_binary(e, minV, maxV):

    result = e*(maxV-minV) + minV

    return result

class NeuralNetwork:
```

```
def __init__(self, layers, activation='tanh'):

    """

    :param layers: A list containing the number of units in each layer.

    Should be at least two values

    :param activation: The activation function to be used. Can be

    "logistic" or "tanh"

    """

    np.random.seed(0)

    if activation == 'logistic':

        self.activation = logistic

        self.activation_deriv = logistic_derivative

    elif activation == 'tanh':

        self.activation = tanh

        self.activation_deriv = tanh_deriv

    self.weights = []

    for i in range(1, len(layers) - 1):

        self.weights.append((2*np.random.random((layers[i - 1] + 1, layers[i]

                            + 1))-1)*2.0)

    self.weights.append((2*np.random.random((layers[i] + 1, layers[i +

                        1]))-1)*2.0)
```

```
def fit(self, X, y, learning_rate=2, epochs=50000):

    X = np.atleast_2d(X)

    temp = np.ones([X.shape[0], X.shape[1]+1])

    temp[:, 0:-1] = X   # adding the bias unit to the input layer

    X = temp  # Create a new X but with an extra bias item

    y = np.array(y)

    for k in range(epochs):

        i = np.random.randint(X.shape[0])

        a = [X[i]]

        for l in range(len(self.weights)):

            a.append(self.activation(np.dot(a[l], self.weights[l])))

        error = y[i] - a[-1]

        deltas = [error * self.activation_deriv(a[-1])]

        for l in range(len(a) - 2, 0, -1): # we need to begin at the second to last layer

            deltas.append(deltas[-1].dot(self.weights[l].T)*self.activation_deriv(a[l]))

        deltas.reverse()

        for i in range(len(self.weights)):

            layer = np.atleast_2d(a[i])

            delta = np.atleast_2d(deltas[i])

            self.weights[i] += learning_rate * layer.T.dot(delta)

def predict(self, x):
```

```
x = np.array(x)

temp = np.ones(x.shape[0]+1)

temp[0:-1] = x

a = temp

for l in range(0, len(self.weights)):

    a = self.activation(np.dot(a, self.weights[l]))

return a
```

Kalman Filter python code

```
#       Kalman       filter       in       Python       adopted       from       http://scipy-
cookbook.readthedocs.io/items/KalmanFiltering.html

import numpy as np

import matplotlib.pyplot as plt

import time



class KalmanFilter:

    def __init__(self,  base_value=24,  iterations=200,  initial_guess=20.0,  posteri_estimate=4.0,
data=[], plot=False):

        # initial parameters

        self.n_iter = iterations  # How many iterations to create test data

        sz = (self.n_iter,)  # size of array

        self.x = base_value  # This is the base value that shall be used to create noisy data. It
is the true value

        if len(data) == 0:

            self.z  =  np.random.normal(self.x,  1,  size=sz)   #  observations  (normal  about  x,
sigma=0.1)

        else:

            self.z = data



        self.Q = 1e-5 # process variance

        # allocate space for arrays

        self.xhat = np.zeros(sz)      # a posteri estimate of x

        self.P = np.zeros(sz)         # a posteri error estimate

        self.xhatminus = np.zeros(sz)  # a priori estimate of x

        self.Pminus = np.zeros(sz)



        # a priori error estimate

        self.K = np.zeros(sz)         # gain or blending factor

        self.R = 2
        # initial guesses
        self.xhat[0] = initial_guess  # Initial estimate
        self.P[0] = posteri_estimate  # Estimate of the error made
        self.plot = plot

    def filter(self):
        start = time.time()
        for k in range(1, self.n_iter):
            # time update
            self.xhatminus[k] = self.xhat[k-1]
            self.Pminus[k] = self.P[k-1]+self.Q

            # measurement update
            self.K[k] = self.Pminus[k]/(self.Pminus[k]+self.R)
            self.xhat[k] = self.xhatminus[k]+self.K[k]*(self.z[k]-self.xhatminus[k])
            self.P[k] = (1-self.K[k])*self.Pminus[k]
        end = time.time()

        print("Took %s seconds" % (time.time() - start))

        if self.plot:
            plt = self.plot_results()
        else:
            plt = None
        return self.z, self.xhat, self.x, plt

    def plot_results(self):
        plt.rcParams['figure.figsize'] = (10, 8)
        plt.figure()
        plt.plot(self.z, 'k+', label='noisy measurements')
```

```
plt.plot(self.xhat, 'b-', label='a posteri estimate')

plt.axhline(self.x, color='g', label='truth value')

plt.legend()

plt.title('Estimate vs. iteration step', fontweight='bold')

plt.xlabel('Iteration')

plt.ylabel('Temperature')


return plt
```

ANN training data

ID	Noisy Data	Actual Data
301	21.5360465755	22.0
300	21.8743646268	21.9595959596
299	21.669130502	21.9191919192
298	20.3472618312	21.8787878788
297	21.7014780494	21.8383838384
296	22.2256235741	21.797979798
295	20.9565583577	21.7575757576
294	22.0599013607	21.7171717172
293	21.9491300859	21.6767676768
292	22.9409266516	21.6363636364
291	22.3309335226	21.595959596
290	22.1939897915	21.5555555556
289	22.0302445641	21.5151515152
288	21.7298237087	21.4747474747
287	20.6579092532	21.4343434343
286	20.9922750258	21.3939393939
285	22.1636777217	21.3535353535
284	20.8706710452	21.3131313131
283	20.9955494271	21.2727272727
282	21.6130661663	21.2323232323
281	20.4117530522	21.1919191919

280	21.0249838384	21.1515151515
279	20.9702444598	21.1111111111
278	20.2699160068	21.0707070707
277	21.6204703317	21.0303030303
276	21.4166578375	20.9898989899
275	21.0459206569	20.9494949495
274	21.1216069379	20.9090909091
273	19.918905036	20.8686868687
272	20.5090086766	20.8282828283
271	21.3497810107	20.7878787879
270	20.2064608189	20.7474747475
269	20.7630025291	20.7070707071
268	20.0499821752	20.6666666667
267	20.8150497613	20.6262626263
266	20.9415470431	20.5858585859
265	19.8863318045	20.5454545455
264	21.1061980615	20.5050505051
263	20.7526920248	20.4646464646
262	19.7889036326	20.4242424242
261	20.4519889727	20.3838383838
260	21.1578233396	20.3434343434
259	20.6117224438	20.303030303
258	20.7146474916	20.2626262626
257	20.0132119971	20.2222222222

256	20.4716284368	20.1818181818
255	21.066358121	20.1414141414
254	19.6675153187	20.101010101
253	19.6303106708	20.0606060606
252	20.596889755	20.0202020202
251	20.1903302374	19.9797979798
250	21.2662359489	19.9393939394
249	20.2367059154	19.898989899
248	19.332725154	19.8585858586
247	19.3498430576	19.8181818182
246	19.5308710538	19.7777777778
245	20.4597874855	19.7373737374
244	19.6524812626	19.696969697
243	18.7240178902	19.6565656566
242	19.4912462044	19.6161616162
241	20.3267468795	19.5757575758
240	20.1333274298	19.5353535354
239	19.6486771259	19.4949494949
238	19.5165451707	19.4545454545
237	19.9189956103	19.4141414141
236	19.1699957128	19.3737373737
235	19.3572561508	19.3333333333
234	19.8612616344	19.2929292929
233	18.5020260599	19.2525252525

232	18.6395996908	19.2121212121
231	19.0873740855	19.1717171717
230	17.953096565	19.1313131313
229	18.9983707054	19.0909090909
228	19.8604632632	19.0505050505
227	18.2329009267	19.0101010101
226	18.221256918	18.9696969697
225	18.8133790121	18.9292929293
224	19.0802205583	18.8888888889
223	19.3186400839	18.8484848485
222	18.197885428	18.8080808081
221	18.6867889298	18.7676767677
220	18.3451264664	18.7272727273
219	18.5655546653	18.6868686869
218	17.9159654641	18.6464646465
217	18.2791599644	18.6060606061
216	18.7308372528	18.5656565657
215	19.3216908231	18.5252525253
214	18.0955224211	18.4848484848
213	18.4608168925	18.4444444444
212	17.7839901225	18.404040404
211	18.8085615061	18.3636363636
210	18.3699122103	18.3232323232
209	17.6350375581	18.2828282828

208	18.3562055601	18.2424242424
207	18.2416391104	18.202020202
206	17.5388401853	18.1616161616
205	17.9137578193	18.1212121212
204	18.2551065627	18.0808080808
203	17.9871842225	18.0404040404
202	18.0885932842	18.0