Faculty of Technology and Engineering Sciences

Motion Capturing Machine Learning

Bluetooth motion sensors

—

Elisabeth Razafimandimby Gangenes

Master's Thesis in Computer Science SHO-6264 - December 2016

Motion Capturing Machine Learning

Elisabeth Razafimandimby Gangenes

December 30, 2016

Abstract

Good and efficient care for our elderly is a major concern in our ever aging population. The elderly's greatest fear is falling to the ground, not being able to get up or get help. Use of Bluetooth motion sensors and appropriate agents can help alert for in the event of a person falling. In this master thesis an experimental sensor rig has been created together with an android application which can capture and save the data from the rig. This data has then been transfered to a feed-forward backpropagating network in order to train it to recogise falls and separate them from five daily activities. We have used three sensors to accomplish this: an accelerometer; a gyroscope; and a magnetometer. Sensory data from these has been given to three separate networks which performs their own prediction before being sent to an overarching network to perform the final prediction. Results show that it is possible to use such a system to a high degree of recognition, however the system is vulnerable to overfitting so care must be taken during training.

Keywords: *artificial intelligence; pattern recognition; accelerometer; machine learning; bluetooth low energy; gyroscope; magnetometer; artificial neural network; deep learning; motion capture; human activities; data analysis; android applications; feed-forward network; backpropagation network;*

Aknowledgements

I would like to thank professor Bernt A. Bremdal for providing his insight and suggestions during the planning and development of this thesis.

Furthermore I would like to thank Asbjørn Danielsen for his invaluable assistance in providing insight and advise on litterature, aswell as giving feedback during development.

I also wish to extend a special thanks to the ten volunteers I had for the experiments performed in this thesis.

Finally I would like to extend my deepest gratitude to everyone who has helped to keep me motivated and inspired during this thesis.

Contents

Chapter 1

Introduction

This paper is the report for a Master's Thesis for the Master of Science program Computer Science (Narvik) - master, ran at UiT / Norges Arktiske Universitet - Campus Narvik in 2016.

Motion capture is the act of recording the movements of people, or objects. It has a broad specturm of applications: military; civilian; entertainment; medical; the possibilities are endless. In this thesis we will examine a medical approach to motion capture and use it to capture and recognise some activities performed daily, and one particular accident case: a fall.

Machine learning, first described in 1959 by Arthur L. Samuel [27], is a field within computer science used to describe the field which "gives computers the ability to learn without being explicitly programmed". The field has evolved from the study of pattern recognition computational learning theory into an area encompassing algorithms capable of learning from and making predictions on data [11]. We will use a machine learning algorithm in order to process and train an agent to see whether a set of data is of a set of someone who is falling or not.

1.1 Background

In the public healthcare system there are several applications where active motion analysis can be useful for monitoring the state of patients. In particular in monitoring elderly patients, who are prone to falling over. According to an article by Christine Gulbrandsen (2009) for forskning.no, elderly who fall over more than twice a year has a 57 % higher fatality rate compared to those who do not [5]. The injuries from these falls are a major health burden, particularily in the long-term care environment and they account for majority of hip and wrist fractures and head injuries, according to a study made by Grisso, Schwarz, Wishner, Weene, Holmes, and Sutton (1990) [7].

It is in society's best interest to develop preventative measures for these falls or ways to detect and alert if a fall has occured. An observational study made by Robinovitch and Feldman (2012) found that a fall is caused when the patient attempts to move the center of gravity before moving their feet [24]. If we can monitor the patient's center of gravity and possibly other limbs, we can predict when a fall is about to occur and if we also can monitor the location of the patient's chest and/or head we can detect if the patient is lying prone after a fall. But for this thesis we will see if it is possible to create a portable, personal capture system using bluetooth sensors and an agent which can analyse and recognise the movements of subjects.

1.2 Existing systems and collaboration

This thesis has been carried out independently, but I have recieved much insight from my supervisor and assistant supervisor. When it comes to fall detection and the like there are several existing solutions and studies done on the subject. Looking at the market there are several existing devices which can detect falls. However looking into reviews of these devices reveal that even the best have a detection rate between 50% and 80% [22] when tested by Top Ten Review. According to what's written on Top Ten Review the norm seems to be a detection rate of 75% for these devices.

1.3 Thesis description

The goal of this project is demonstrate movement capturing and recognition by means of bluetooth sensors. This implies that an experimental rig will be created that connect one or more bluetooth devices which can capture movement patterns of a person and develop an agent which can learn to recognise these movements. A smart phone will be used to record the data, while a desktop program will perform the machine learning. Examine the learning process.

1.4 Limitations

1.4.1 The Data Set

The work and study performed in this thesis is based around a limited dataset with preset exercises and supervised learning. The participants of the exercises described in Section 3.2 are aged between 20 and 50 years old and are of both genders, but the age range is not represented within the genders. Because the exercises are planned the data sets will not account for other activities and variations within these activities outside of the participant's performance. There are also limitations on the sensor selected for the data collection: it is not designed for outdoor and extreme enviromental conditions, therefore the data sets will be limited to indoor movements.

When compiling all the data sets together we get around 5768 data sets, where

710 of them are of the participants falling.

1.4.2 The learning method

Several algorithms exists for supervised learning, and we will explore some in 2.4, but only one has been selected for further study in this thesis.

1.4.3 Software limitations

The software I have created during this thesis is built as suit-for-purpose. It currently does not support any other type of data and will fail to read and interpret data sets that does not follow the data structures described in 3.3.1.

Furthermore I have not had the opportunity to implement saving functionality for the machine learning software. This may make it difficult to recreate my results, even if give the refined network settings are given, due to the randomised starting positions in the machine learning system.

Lastly the system operates on a “fall detected” and “no fall detected” on the outputs.

Chapter 2

Theory

2.1 Motion Capture

Motion capture is the act of capturing motions performed by an actor. In the context of this thesis, this term is usually used to animate characters in computer games or movies, but it has other more unexplored applications aswell. There are several ways to do this and there are three main categories of motion capture systems: inside in, inside out and outside in. Most of the material in this section can be found in [31].

2.1.1 Inside In

There are three primary ways to perform inside in based motion capture: electromechanical suites, optical fiber and accelerometer based systems. Inside In's benefits are that they are very portable and can mostly be used anywhere. Accuracy in these systems vary greatly, but some are also heavily restricted in the type of motions they can capture. More on these restrictions in the paragraphs about the systems. Examples of all these systems can be seen in figure 2.1.

Electromechanical suites

Electromechanical suites typically involve exo-skeletons or armatures worn over the subject. It is outfitted with rods connected with potentiometers which record the analog voltage changes and convert it into digital values. This gives the user a very accurate capture of motions in real-time. The biggest problem with these suites is that they're very restrictive, generally not allowing full range of motions and they need to be customised for the user to match the body proportions. They also do not allow for global positions.

Optical fiber

Optical fiber are the most restrictive type of the Inside In systems. It is typically used for data gloves and work by running fiber-optic sensors along the seams, in the case of gloves along the fingers. The bent fibers attenuates light and this light is converted into measurement. This gives real-time and inexpensive measurement, but are typically only used for hands and are of poor quality. Hence this method is best suited for capturing gestures.

Accelerometer based systems

Accelerometer based systems works by having internal gyroscopes and accelerometers to measure acceleration and orientation. Typically built into suits, they are inexpensive and work in real-time, however they do not provide global position and the suit can be restrictive. Some may also contain ultrasonic sensors to measure distance, but from what I have found this particular feature isn't as common for personal use devices.

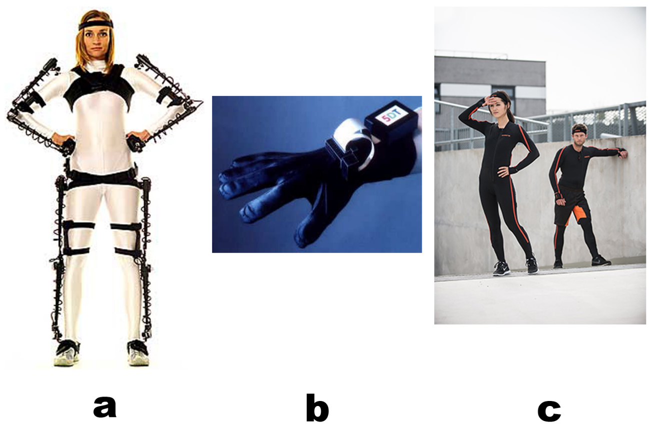


Figure 2.1: a) Gypso 6 exosceleton from Animazoo. Source: Wordpress b) 5DT - 5 sensor Data Glove. Source: Metamotion c) XSens Inertia Suit. Source: XSens

2.1.2 Inside Out

Inside Out typically has three main ways to capture motions: Electromagnetic, Semi-passive and Optical capture. The portability of these systems vary, as does the systems' accuracy.

Electromagnetic

Electromagnetic systems work by putting external transmitters to establish magnetic fields in space. Sensors are then used to capture the transmitter's positions and orientations. The data from these sensors are then transmitted back either wirelessly or across wire. This method captures motions in real-time, however it has limited range, is vulnerable to interference from metal objects in the space and the system is quite expensive to maintain and use.

Semi-passive

These systems typically work by either using Multi-LED IR projectors in the environment which emit spatially varying patterns. Photo-sensitive markers tags on the subjects decote the signals and estimates their position. This gives them real-time and high speed tracking at a low cost. However due to the nature of IR they can struggle with the accuracy and they do no support global position. Furthermore this method is rather invasive on the subject due to the number of markers which needs to be attached to the subject. It also makes portability an issue outside of one's home, as we will need to bring the IR projectors with us.

Optical

The optical inside-out system involves use of several high framerate cameras in order to capture the environment around the user. It then uses a method called Scale Invariant Features Transformation, SIFT [16], to extract features from the environment to triangulate the cameras' positions. This yields a very portable method of motion capture which can give global position. However for personal use this is a very expensive method, as each camera has a moderate cost, but the user requires several of them in order to get useable data. The data given, however, is not of high quality as it reconstructs an environment based on the features in order to find it's position. This also makes the system very slow.

2.1.3 Outside In

The most common method of outside in motion capture is the use of optical systems and markers, but there also exists marker-less systems. This involves placing markers on the actor's body to track them with two or more cameras and triangulating the markers' positions in a 3D environment using these cameras, see

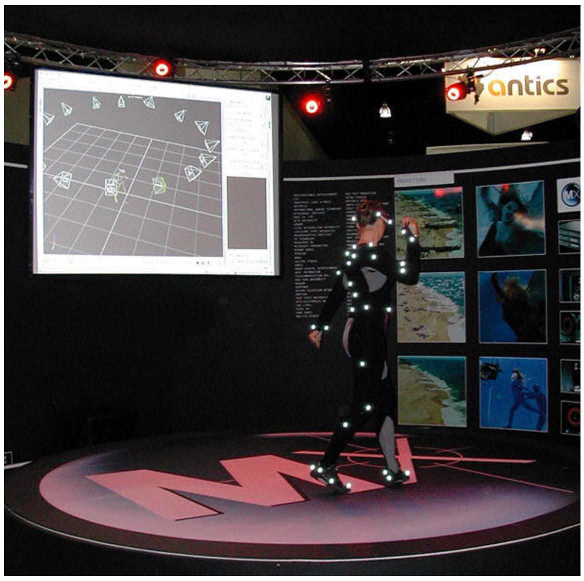


Figure 2.2: A dancer wearing a suit with passive markers used in an optical motion capture system. Source: Wikipedia

figure 2.2. These markers can be as simple as retroreflective surfaces to reflect light generated close to the capturing cameras, called passive markers, to flashing LEDs and accompanying software to recognise their relative positions, called active markers. This method offers the highest possible precision, but it's not very portable.

Passive markers

Passive markers have the weakness of marker swap occuring. This is when the system gets confused by the relative position of one or more markers, placing any one marker in multiple positions. To combat this effect a system typically has upwards to 48 cameras. The advantage of passive markers is that the user is not required to wear additional wires or electronic systems, which gives them more freedom of movement. The framerate of the capturing system also affect the accuracy of the sample data. It is typical to capture at a rate of 120 to 160 on movement, but having as high as 10000 fps when capturing smaller regions of interest, such as capturing facial expressions.

Active markers

Active markers typically include illuminating one LED at a time at a rapid pace and identifying their positions with software. This method has a higher accuracy potential when compared to the passive markers, down to 0.1 milimeters [14]. The downside is that these markers take more space and may not be suitable for minute measurements like capturing facial expressions.

2.1.4 Discussion

After looking into the varying ways of capturing motion we can exclude both Inside Out and Outside In methods, they have several problems which will make them unsuitable for our use. When it comes to Inside In systems we can discard an electromagnetical suite as they do not allow for the freedom of movement needed for personal use, and they need to be customised for the subjects. The optical fiber is discarded on it's limitations aswell, as we need to capture more than just hand gestures.

This leaves the accelerometer based system. They are typically built into suits, but if we can find sensors which are small enough to strap onto the actor's body in various places we can create a system which allows us to effectively capture the movements. The device will need to be small enough to not restrict the actor's motions, yet sensitive enough to also register minute movements.

2.2 Bluetooth Low Energy, BLE Technology

Bluetooth Low Energy, also marketed as Bluetooth Smart, is a wireless personal area network system created and marketed by Bluetooth Special Interest Group. It was introduced under the name Wibree in 2006 [17] by Nokia. It hold certain characteristics that differ from regular Bluetooth such as a lower transferral rate, 1 Mbps as opposed to Bluetooth's 3 Mbps and the chips being smaller. The most prominent feature however is that BLE remains in sleep mode except for when a connection is initiated [12]. This contributes to the increased battery expectancy when using BLE over Bluetooth. It was later introduced as part of Bluetooth 4.0.

One noteworthy attribute when developing with BLE technology is that, unlike normal Bluetooh, every device always functions as a server, while the interracting device is always a client. This is an important distinction when working with multiple devices on a single application, which might be the case with this project (see 3.1). This is due to the GATT/ATT relationship which is a standard when working with BLE profiles [4].

2.2.1 Attribute Protocol, ATT, and Generic Attribute Profile, GATT

The Attribute Protocol, ATT, is a low level wire application protocol that defines how to transfer a unit of data. An attribute is composed of three elements: a 16-bit handle, an standardised 128-bit format string called a Universally Unique Identifier (UUID) [4] which defines the attribute type and a value of a certain length [20].

The Generic Attribute Profile, GATT, is built on top of the Attribute Protocol, and it dictates how the ATT is employed in service composition [20]. It is a base profile for all top-level LE profiles and details how various ATT attributes are grouped together to form meaningful services. The GATT is what serves as the unit's server and is what connects the BLE device to whatever client application is using the unit.

2.3 Android

Android is an operating system created by Google for use on mobile devices, particularly on touchscreen devices. It's based on the Linux kernel. When developing to the system it is important to note the API levels. These levels decide the compatibility of your application with various devices. As a rule of thumb, it's best to utilise the lowest possible level to have the broadest possible compatibility.

2.3.1 Android 4.3, API Level 18

BLE was first introduced into Android devices with version 4.3. So this will be the minimum application level.

2.4 Machine learning methods

This section will look into the three supervised machine learning algorithms evaluated for this thesis. There has been several other publications on this topic, and their findings and analysis will be referenced throughout the section.

2.4.1 Support Vector Machines

Support Vector Machines, SVM, is a supervised learning model. This means that the system is trained by giving examples at which to build it's recognition. It constructs a hyperplane, or a set of hyperplanes, in a high- or infinite-dimensional space. This plane is then used to divide the data into clusters

In standard formulation the SVMs are linear classifiers, however it is possible to extend the SVM by using a technique called Kernel Trick, or Kernel Method, to achieve non-linear classification [3]. This method involves projecting the data from the original data space to a high dimensional feature space by using a non-linear kernel function. Linear separation can then be achieved by using Cover's theorem [3, ref: 66]. This method is however time-consuming and may not be appropriate to use with large quantities of data. The algorithm also only has one output.

According to [2] the SVM is the best second best performing algorithm for inertia based sensors, however it also has the slowest training- and testing times.

2.4.2 Neural Networking

An artificial neural network is a machine learing algorithm inspired by the biological neural networks, in particular the brain, which is used to estimate or approximate functions [19]. First concieved in 1943 by Warren McCulloch [21], it can process a large number of inputs, that are generally unknown, into an output. They are typically secified using three attributes: the architechture which specifies the variables involved in the network and their topological relationships; the activity rule which defines how the activities of neurons change in response to each other, typically by a weight parameter; and the learning rule which specifies the rate at which the weights of the system changes. Allende, Moraga and Salas' article, Artificial Neural Networks in Time Series Forecasting: A Comparative Analysis [26, c. 3], has an excellent description of the workings of a neural network, it's research and it's applications.

A neural network has three essential features: the neurons, can also be called nodes or perceptrons, which are the basic processing elements of the network; the network architecture which describes how these nodes are connected; and the training algorithm used to find the values of the network parameters for performing a task. The neurons are organised into three layers: input; hidden which may have more than one layer within it; and output. The number of input units is determined by the application of the network, eg. data points to be processed to an output. Once again Allende, Moraga and Salas' article provides an excellent rundown on the elements of an artificial neural network and it's mathematical construction [26, c. 3.1].

A problem with neural networks are that they are somewhat of a black box. It is hard to see how they're solving a problem, and because if this it is difficult to troubleshoot them. They are also not probabilistic, so it's hard to predict how an epoch will turn out. Furthermore they cannot be retrained with additional data. If one wants to add data to the network one needs to retrain the whole thing from the bottom.

According to [2] the neural network is the worst performing algorithm for inertia based sensors looked into for this study, scored at number 5 of 7 in their study. It also has the second slowest training time, however the testing time for the neural network is among the best in [2], and for this thesis it is the fastes for testing, which is crucial in the public healthcare system.

2.4.3 k-Nearest Neighbour

The k-Nearest Neighbours, k-NN, algorithm is a non-parametric method for classification and regression [1]. In classification the output is a class membership where the input object is assigned to a class by a majority vote by it's neighbours, with the object assigned to the class most common among it's k nearest neighbours. If k=1 then the object is assigned to the class of that single nearest neighbour. In the case of regression the output is the property value of the object and this value is the average of values of it's nearest k neighbours. The advantages to the algorithm is it's robustness to noisy training data and it's effectiveness with large training data. The main drawback of this algorithm is the complexity in searching for the nearest neighbours for each sample and the high computational costs because of this complexity. Other drawbacks includes the need to determine the parameter for k and that distance based learning is not clear on what type of distance to use; do we use all attributes or just some?

[2] and [3] finds the k-NN algorithm as the best algorithm for recognizing human activities with inertia based sensors at a staggering 96,26% accuracy. The training and testing time for the algorithm is however mediocre [2, t. 4]. Both training and testing times are listed together t 3.4 seconds, which puts it as the fastest training algorithm in this thesis, but as the second at training.

2.4.4 Discussion

When looking at the study made by Attal, Hohammed, Debabrishvili, Chamroukhi, Oukhellou and Amirat on movement recognition when using raw data, we can see that the k-Nearest Neighbour is the best algorithm for supervised learning. It has a high precision rate of 94% [3, Table. 3, 4]. Use of k-Nearest Neighbour is also supported in [**?**], where it is the algorithm with the highest accuracy of 96%.

After looking into the algorithms and discussing them with my supervisor, we agreed upon going forward with the Neural Network algorithm for this thesis. We will look more into the spesifics of the neural network in 3.4. The publications I've studies also does not seem to utilise a deeper learning network, nor do they seem to be using more than one network.

At the end of the thesis I believe it might have been more beneficial to have chosen SVM or k-NN like [2] suggests for recognition, however it is worth noting that according to [2] both SVM and k-NN is far slower than the Neural Network once trained, with testing times of 5.95 seconds and 3.4 seconds for the SVM and k-NN respectively against the 0.02 of the neural network. This difference in time can be crucial in the public health sector.

Chapter 3

Methods

3.1 Experimental Rig

The experimental rig consists of three connecting modules: the sensors, the capturing device and the learning agent, see figure 3.1 on page 23. The sensors will consist of one or more bluetooth devices which the capturing device will read and save data from. From there the saved data files will be transmitted or extracted, be it automatically or manually, to the learning agent for analysis.

Because the project demands capturing movement at any time, we will use an Inside In approach to capture the motions. This is due to the portability and need to capture motions outside a pre-setup studio.

3.1.1 TI SimpleLink Multi-Standard SensorTag, CC2650 wireless MCU

In recommendation from assistant supervisor, Asbjørn Danielsen, I have decided to make use of the Texas Instruments SimpleLink Multi-Standard SensorTag CC2650 wireless MCU, hencefourth refered to as the CC2650, as the motion capturing sensor. It is a portable sensor, being only a few centimeters in length (see figure 3.2), which can easily be attached upon a subject's body and operates through Blue-

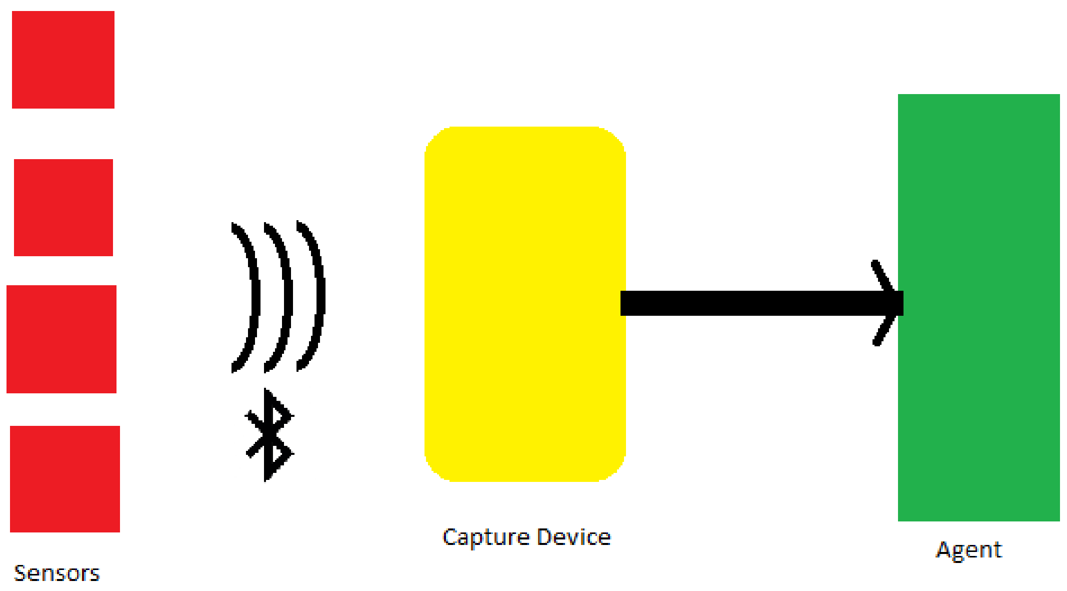


Figure 3.1: Mockup of experimental rig

tooth Low Energy, see section 2.2. The CC2650 features several different sensors, but for the purposes of this project I predict only two of them will be used: the movement sensor and barometric pressure sensor. One thing to note is that the CC2650 is not designed for outdoor and extreme enviromental conditions, therefore this study will limit the test cases to movement indoors. They also need to be properly secured, as the unit currently only has a small handle which can be used to fasten the sensor. A distinct advantage is that the CC2650 has readily available software and source code available, so I do not have to develop a protocol to capture and interpret the data from the sensor.

Movement Sensors

The CC2650 features the MPU-9250 created by InvenSense [9] which is the company's second generation 9-axis motion tracking devies. It is a System in Package which combines two chips: the MPU-6500, which has a 3-axis gyroscope, an 3-

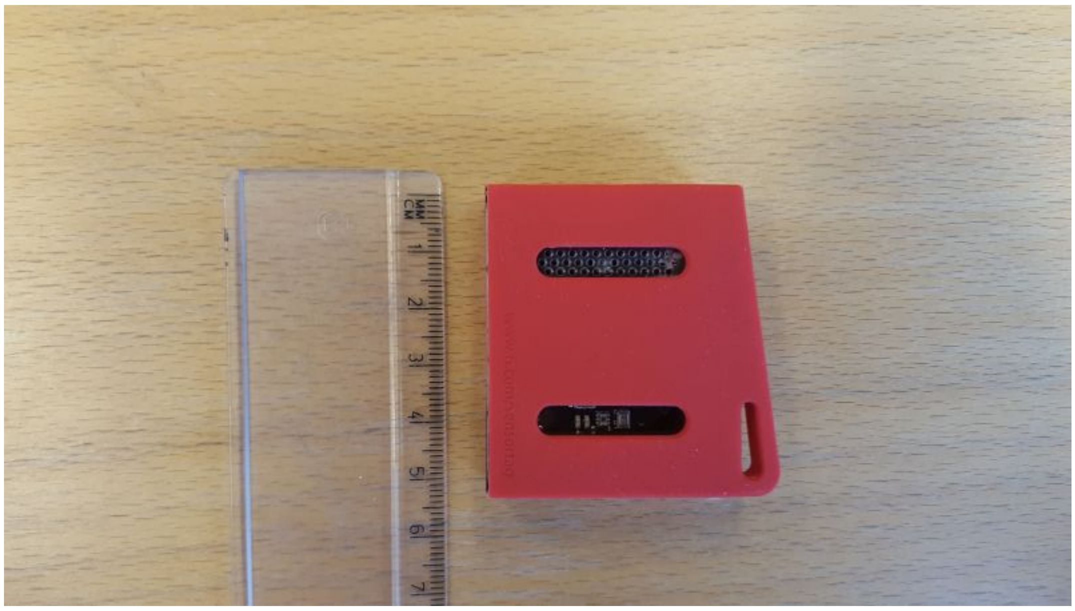
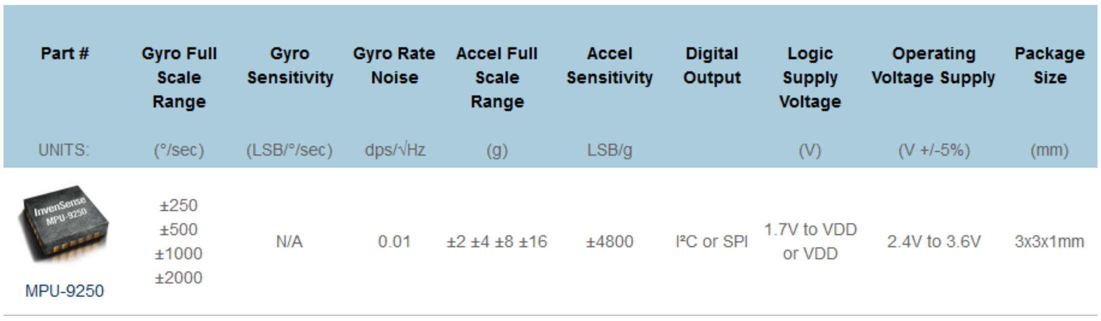


Figure 3.2: TI SimpleLink Multi-Standard SensorTag, CC2650 wireless MCU

axis accelerometer, and an Digital Motion Processor (DMP) capable of processing MotionFusion algorithms; and the AK8963, a 3-axis digital compass. The sensor outputs raw data which consists of nine 16-bit signed values, one for each axis. The device is compliant with Android version 4.1, Jelly Bean [10], and supports new low-power DMP capabilities that offload the host processor to reduce energy consumption and simplify application development. Rough spesifications of the sensor's capabilities, dimensions and power demand can be seen in Figure 3.3.

Pressure Sensor

The CC2650 features the BMP280 created by Bosch Sensortec [9] which is an absolute barometric pressure sensor. According to Bosch Sensortec the sensor can be used for indoor navigation for detecting floors and elevators [29]. This functionality will let us analyse the movement of the subject's movements in the y-axis to determine if they are moving in stairs, elevators or other forms of elevation,



Part #	Gyro Full Scale Range	Gyro Sensitivity	Gyro Rate Noise	Accel Full Scale Range	Accel Sensitivity	Digital Output	Logic Supply Voltage	Operating Voltage Supply	Package Size
UNITS:	(°/sec)	(LSB/°/sec)	dps/√Hz	(g)	LSB/g		(V)	(V +/-5%)	(mm)
MPU-9250	±250 ±500 ±1000 ±2000	N/A	0.01	±2 ±4 ±8 ±16	±4800	I²C or SPI	1.7V to VDD or VDD	2.4V to 3.6V	3x3x1mm

Figure 3.3: MPU-9250 spesification table. Source: Invensense

which can be useful for telling where the they are located. Tests will determine if the sensor is sensitive enough to determine whether the subject is lying on the floor or not, but the technical data suggests the data resolution seems to be with a precision of less than 10 cm [29], with a relative absolute accuracy of within 1 meter, so this functionality should be possible to implement.

3.1.2 The sensor rig

The sensor rig is a simple construction consisting of a single CC2650 unit, a belt, a hairtie and duct tape to secure the sensor to the belt, as shown in Figure 3.4. The belt is strapped around the waist of the subject where the hip bone meets the spine, as that gives the highest single sensor recognition [3, Table. 1].

3.1.3 Capture Device

We will be using an Samsung Galaxy S6 for the purposes of capturing the data broadcasted by the CC2650. Texas Instruments has a capturing software for android devices, but it only accomodates one sensor at a time and does not save the data. Texas Instruments has the source code for this software available on their Gitorious server and appropriate modifications will be done to it to allow saving of data and possibly use of multiple sensors. The capturing device will not be

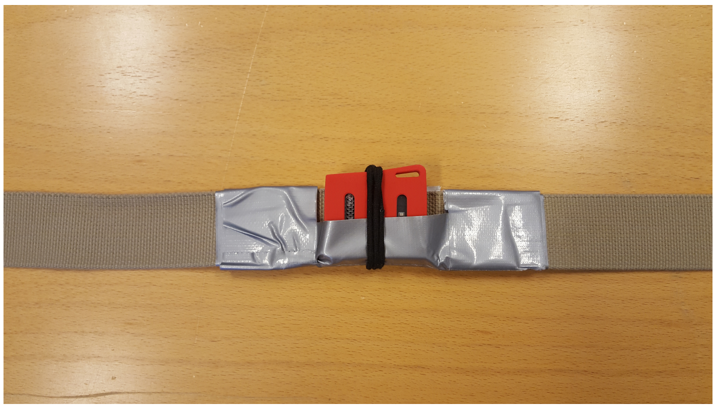


Figure 3.4: Experimental sensor rig

performing any analysis of the data.

Preliminary study suggests that a Raspberry Pi might be more suitable as a capture device for commercial use, but for the purpose of this study a smartphone is more suitable due to Texas Instrument's ready to use software.

3.1.4 Learning Agent

In order to process the data we will create a machine learning algorithm and a pre-processing method to interpret the data of each sensor in conjunction with what it's company sensors detect, as stated by the thesis description. This will be a desktop software and will not be running in real-time with the sensors for the purposes of this project.

The algorithm selected for this agent is the Artificial Neural Network, the workings of which we will examine in-depth in 3.4.

The data is pre-processed through a parser which will separate the various sensor data and organise them into data sets. During this processing the barometric data is currently filtered out and is not included into the data sets. The reasons for this will be examined and discussed in 4.1.

3.2 The exercies

To obtain a varied data set I have employed a small group of volunteers that will perform some simple exercies with the CC2650 SensorTags attached to a belt which will be strapped around the lower back. The participants will be given no spesific instruction on how to perform the actions and they will be free to perform them however they like. To obtain a suitable amount of data I have opted to have a sample rate of 16 observations per second, however a different number may be found to be more appropriate later. The participants will also perform each action three times in three separate sets of data.

The exercises are as follows:

1. *Walking:* A simple walking exerices will be performed over the course of 10 seconds.

2. *Fall onto a falling mat:* Participants will fall forward onto a large mat. A study done by Xingou Yu for the Institute for Infocomm Research puts falls into four types: fall from sleeping (bed); fall from sitting (chair); fall from standing or walking on the floor; and fall from standing on supports such as ladders [34]. For this we will be focusing just on the third type.

3. *Walking up stairs:* The participant will walk up a normal, one floor set of stairs.

4. *Sitting down into a chair at a table:* A chair will be placed by a table for the participant to sit down into. This will involve moving the chair from the table and sitting down.

5. *Picking an item up from the floor, sitting:* An item will be placed on the floor for the participant to pick up. The participant will start from a sitting position and return to the sitting position.

6. *Picking an item up from the floor, standing:* An item will be placed on the floor for the participant to pick up. The participant will start from a standing position and return to the standing position.

3.2.1 Participant Survey

The participants of this study are annonomys, however a small survey of the participating subjects will be held where we note the participant's gender, age and height. In the initial analysis of the data this survey's information will not be utilised, however some of the information found in it may be relevant in further study of the subject's motions.

3.3 TI Sensor Tag android application - Modified BLE Sensor Tag (capture device)

The Texas Instruments provides a ready to use software for capturing the data from it's sensor tags, however the application does not support saving this data to a file and it only supports one data sensor at a time. The source code for this software was found on Texas Instrument's own Gitorious repository [8].

This software was then modified to exclude the sensors we do not use, aswell as functionality to record and save the data recorded from the sensors.

3.3.1 Data structure

The data given by the sensor is structured in the Data class. This class contains: an array containing said data; an array containing a list of enumerators to differentiate the data, in case of different update frequencies on sensors which may be useful for the barometric sensor; and a integer timestamp which dentoes the time at which

Table 3.1: Structure of the data set

Sensor	Id	Data Type	Description	Surfix
Accelerometer	0	Accel_X	Accelerometer data in X direction	AX
	1	Accel_Y	Accelerometer data in Y direction	AY
	2	Accel_Z	Accelerometer data in Z direction	AZ
Gyroscope	3	Gyro_X	Gyroscope rotating on X axis	GX
	4	Gyro_Y	Gyroscope rotating on Y axis	GY
	5	Gyro_Z	Gyroscope rotating on Z axis	GZ
Magnometer	6	Mag_X	Orientation X	CX
	7	Mag_Y	Orientation Y	CY
	8	Mag_Z	Orientation Z	CZ
Barometer	9	Bar	Barometric pressure value	BA

this data is recorded as marked by Unix Timestamp.

Furthermore the data class contains functions which will allow additional data points to be inserted. Because a data object can contain a varying amount of data the structure is standardised to structure seen in Table 3.1, in addition to the timestamp. Should an entry be missing from the data set or be empty it will be denoted in the entry as ND. This destinction is important as the updates from the CC2650 are not syncronised, meaning data from the movement detecting sensors will not contain data from the barometer and vice versa.

I had originally intended to have a sample rate of 16 samples per second, however the highest possible sample rate was 10 per second, which was a hard coded limit put into the original software.

3.3.2 Saved data naming convention

The data from the exercise will be saved into separate files for each activity and iteration. To categorise these data files they will follow a naming convention *XX_Y_Z.csv*, where: XX denotes the participant number; Y denotes the activity, as described in 3.2; and Z denotes the interation.

3.4 The machine learning agent, artificial neural network

3.4.1 Overview

The learning agent is a multilayered artificial neural network with two primary layers: one layer which process the raw sensor data and a layer which process the output from the previous layer. Furthermore the first layer consists of three separate neural networks, one for each sensor in use. During the experiments I discovered that the barometric sensor was not functioning properly or was not precise enough, therefore it is being excluded from the network.

3.4.2 The perceptron

The artificial perceptron is the basic building block of a artificial neural network. It is also called an artificial neuron, or in some cases a node, and it works as a classifier that can map an input vector x_n of real or boolean values and passes it through an activation function. Figure 3.5 shows this function as a simple step function, but there are several ways to do this.

For example the step function in Figure 3.5 could be:

$$f(x) = \begin{cases} 1 \text{ if} \sum_{j=1}^{n} w_j x_j + b > 0 \\ \quad 0 \text{ otherwise} \end{cases} \tag{3.1}$$

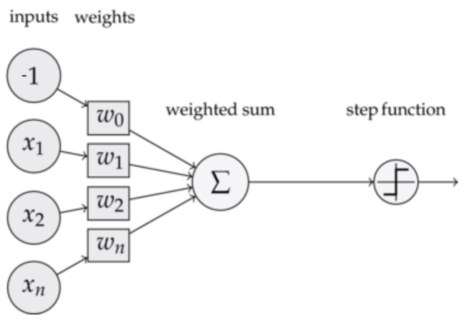


Figure 3.5: Illustration of a perceptron with inputs, weights, a bias and an output.

where the sum of the real-valued weights, w, multiplied with the n number of inputs, x, needs to be greater than the bias, $|b|$, in order to give the output of 1, or the activation of the perceptron.

If the perceptron is part of a large network, a sigmoid transfer function may be put before or in place of the step function.

Training the perceptron

In order to train the perceptron we utilise an algorithm [15]:

Definitions:

$$\begin{aligned} D = (x_1, d_1), ...(x_s, d_s) &\text{ is the training set of } s \text{ samples} \\ x_j &\text{ is the n-dimensional input vector} \\ d_j &\text{ is the target output for } x_j \\ w_i(t) &\text{ is the weight for for the i'th node at time } t \\ y_j(t) &\text{ is the output calculated at time } t \end{aligned} \tag{3.2}$$

The error, or cost, fuction for a training is defined as:

$$E = \frac{1}{s} \sum_{j}^{s} [d_j - y_j(t)] \tag{3.3}$$

In order to train the perceptron we need to update the weights until the inputs is close to the desired output for all examples in the training set D. The error is calculated between the actual output and the target output using Equation 3.3, and can be thought of as an gradient descent [19].

The perceptron is initialised and updated utilizing the following steps:

1. Initialise the weights $w_i(0)$ with small random values.
2. For each sample D_j do the following operations for each input x_j and the target output d_j
   (a) Calculate the output from the perceptron by:
   $$y_j(t) = f[w(t) \cdot x_j] = f[\sum_n w_n(t)x_{j,n}] \tag{3.4}$$
   (b) Update the weights by
   $$w_i(t+1) = f[w(t) \cdot x_j] = f[\sum_n w_n(t)x_{j,n}] \tag{3.5}$$
   for all nodes $0 \leq i \leq b$.
3. Repeat from 2. until a sufficiently low error has been found or until the number of iterations, or epochs, have reached a set limit.

Limitations of a single perceptron

While it is a simple construction, the perceptron can find a solution to several datasets provided they are linearly separable. The training set D is linearly separable if the data can be separated by a hyperplane.

The typical example given when explaining this with a perceptron are the logical expressions “AND”, “OR” and “XOR”. Of these expressions “AND” and

"OR" are linearly separable and thus can be solved by a single perceptron. "XOR" on the other hand is not linearly separable, and thus cannot be learned by a single perceptron.

In order to learn the nonlinear expressions we build layers of perceptrons into a network where the output of one layer is fed into the next layer as inputs through a logistic function. Typically this is the sigmoid function, explained below, and through this method we can approximate solutions to nonlinear sets. This construction is called a "feed forward" neural network, as it feeds the layer outputs forward to the next layer.

3.4.3 Feed forward and backpropagation network

This project implements an artificial neural network, or a feed forward and backpropagation network, to analyse the data sets. Like a single perceptron, it requires a training set with desired outcomes in order to learn. The error is calculated in a similar way, however unlike the single perceptron the error is backpropagated through the neurons in each layer of connected perceptrons. The weights are also updated in a different way by their influence on the error. Much of the implementations have been derived from [18, 19]

Sigmoid neuron

When building a of perceptrons into a neural network, a sigmoid function is typically used as outputs to give a representation of the activation output for the neuron. The sigmoid transfer output is given as:

$$y = \sigma(t) = \frac{1}{1 + e^{-t}} \tag{3.6}$$

where:

$$t = \sum_{j=1}^{n} w_j x_j + b \tag{3.7}$$

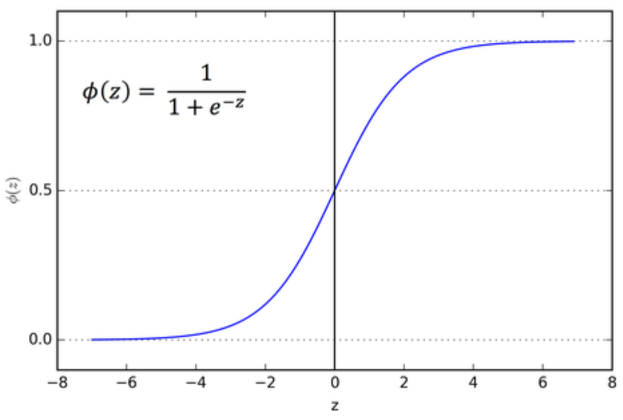


Figure 3.6: Sigmoid fuction

with x_j as the inputs for the node, w_j as weights of the inputs and the bias b. The sigmoid function is continous and differentiable by:

$$\frac{dy}{dt} = y(t)[1 - y(t)] \tag{3.8}$$

This differential is used when updating the weights and it is far less intensive computationally, which is why the sigmoid is often used in backpropagation networks. Figure 3.6 illustrates how the sigmoid function behaves.

Training a neural network

In order to train a neural network certain steps are required. For the purposes of explaining we will look at a simple neural network with a simple input, X; single hidden layer, H; and a single output neuron, O, see Figure 3.7. The output, O, for this network is defined by:

$$O = y(w_1x_1 + w_2x_2 + ... + w_nx_n + b) \tag{3.9}$$

where w is the weight vetor, x is the input vector and b is the bias.

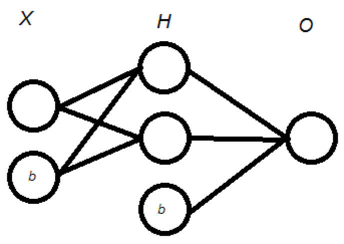


Figure 3.7: A simple neural network with one input, one hidden layer and one output.

Definitions:

$$\begin{aligned}
x_{i,l} &\text{ is the input vector of the } i\text{'th neuron in the } l\text{'th layer}\\
b_{i,l} &\text{ is the bias of the } i\text{'th neuron in the } l\text{'th layer}\\
o_{i,l} &\text{ is the output of the } i\text{'th neuron in the } l\text{'th layer}\\
w^l_{i,j} &\text{ is the weight connecting to neuron } j \text{ in the previous}\\
&\text{ layer to neuron } i \text{ in the current layer } l\\
y(x) &\text{ is the transfer function}\\
d_i &\text{ is the target output for the } i\text{'th neuron in the output layer.}\\
\delta_{i,l} &\text{ is the delta of the } i\text{`th neuron in the } l\text{'th layer}\\
\alpha &\text{ is the learning rate}\\
\theta &\text{ is the momentum}
\end{aligned} \tag{3.10}$$

Unlike the perceptron, the neural network typically uses a different error function. The most normal function for this is the Mean Squared Error, defined as:

$$E = \frac{1}{2}\sum_{i\epsilon O}^{s}(o_i - d_i)^2 \tag{3.11}$$

In order to minimise the error of each output we need to calculate a Δ value. This delta determines the rate of change in the error per weight in the current layer to the previous layer. This value is then used to adjust the weights backwards through the network through backpropagation.

The deltas are calculated in the following way:

- For the output layer, $l = O$:

$$\begin{aligned}\Delta_{i,j}^{O} &= \frac{\partial E}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}} \frac{1}{2} \sum_{i \epsilon O} (o_{i,O} - d_{i,O})^2 = \frac{\partial}{\partial w_{i,j}} o_{i,O}\ (o_{i,O} - d_{i,O}) \\ &= \frac{\partial}{\partial w_{i,j}} y(x_i)\ (o_{i,O} - d_{i,O}) = \frac{\partial}{\partial w_{i,j}} x_{i,O}\ y(x_{i,O})(1 - y(x_{i,O}))(o_{i,O} - d_{i,O}) \\ &= o_{i,O}(o_{i,O} - d_{i,O})(1 - o_{i,O})(o_{i,H})\end{aligned} \tag{3.12}$$

  As the expression is dependant on the previous layer we can simplify the expression as:

$$\delta_{i,O} = o_{i,O}(o_{i,O} - d_{i,O})(1 - o_{i,O}) \tag{3.13}$$

$$\Delta_{i,j}^{O} = \frac{\partial E}{\partial w_{i,j}} = \delta_{i,O}\ o_{i,H} \tag{3.14}$$

- Likewise for the hidden layer, $l = H$:

$$\Delta_{i,j}^{H} = \frac{\partial E}{\partial w_{i,j}} = o_{i,H}(1 - o_{i,H})(o_{i,X}) \sum_{i \epsilon O} (o_{i,O} - d_{i,O}) o_{i,O} (1 - o_{i,O}) w_{i,j}^{O} \tag{3.15}$$

  We can here substitute $\sum_{i \epsilon O} o_{i,O}(o_{i,O} - d_{i,O})(1 - o_{i,O})$, as we can see in Equation 3.13:

$$\Delta_{i,j}^{H} = o_{i,H}(1 - o_{i,H})(o_{i,X}) \sum_{i \epsilon O} \delta_{i,O} w_{i,j}^{O} \tag{3.16}$$

  Which gives the simplifications:

$$\delta_{i,H} = o_{i,H}(1 - o_{i,H}) \sum_{i \epsilon O} \delta_{i,O} w_{i,j}^{O} \tag{3.17}$$

$$\Delta_{i,j}^{H} = \frac{\partial E}{\partial w_{i,j}} = o_{i,X}\ \delta_{i,H} \tag{3.18}$$

  As we can see in the Equation 3.15, the starting derivatives in Eqaution 3.13 propagating through the hidden layer. Should there be more than one hidden layer, the X through Equations 3.15- 3.18 will be replaced with that layer number. When we finally go from the hidden layer to the input layer, the expression will likewise have grown to include all the deltas, like in Equation 3.17.

The biases output a constant value of 1 and they have no inputs, and serve to shift the activation functions of the neurons.

Now that we know how to calculate the deltas, we initialise the network and use the deltas to update the weights by the following steps:

1. Initialise the weights $w^l_{i,j}$ with random values.

2. Run the feed forward function once to get an initial output.

3. For each $o_{i,O}$ calculate $\delta_{i,O}$.

4. For each $o_{i,H}$ calculate $\delta_{i,H}$.

5. Update the weights by:

$$\Delta^l_{i,j} = -\alpha \cdot \delta_{i,l} \cdot o_{i,l-1} \tag{3.19}$$

   which gives the new weight:

$$w^l_{i,j} \overset{+}{=} \Delta w^l_{i,j} \tag{3.20}$$

6. Repeat from 2. until a sufficiently low error has been found or until the number of iterations, or epochs, have reached a set limit.

The learning rate, α is typically $0.0001 \leq \alpha \leq 1$, however any number can be selected. [18, c. 3] has a good illustration of the effects of the learning rate where the effects of the learning rates $\alpha = 0.025$, $\alpha = 0.25$ and $\alpha = 2.5$ and what it does to the training. The summary of the fuction of the learning rate is that it controls the step size within the error gradient descent, to make sure we do not overshoot, but also have an efficient learning.

[18, c. 3] also has a good method of determining the value of α when refining one's network. It also describes changing the learning rate as the network improves, however this has not been implemented with this system.

Introducing the momentum term

Because the errors calculated are like a surface, one problem that can occur is a local minimum in the error surface. This will cause the training to get stuck, as it will not find a better low error. In order to combat this we introduce a momentum-term to the deltas updating in order to push the system out of any potential local minimums. This is done by using the momentum to add a fraction of the previous delta to the current iteration.

This implementation gives a new equation for $\Delta^l_{i,j}$:

$$\Delta^l_{i,j} = \Delta^l_{i,j}(t) = -\alpha \cdot \delta_{i,l} \cdot o_{i,l-1} + \theta \cdot \Delta^l_{i,j}(t-1) \tag{3.21}$$

where t is the current epoch, or iteration.

Reducing overtraining

One problem we must respect when it comes to training any sort of machine learning agent is overtraining the system. Figure 3.8 illustrates a case of overfitting where noisy linear data has been fitted perfectly with a polynominal function. According to [6] "overfitting is the use of models or procedures that violate parsimony - that is, that include more terms than are necessary or use more complicated approaches than are necessary". In other words: overfitting typically occurs when the system is overly complex compared to the problem being analysed. In the case of a neural network this could be having too many neurons within the system,

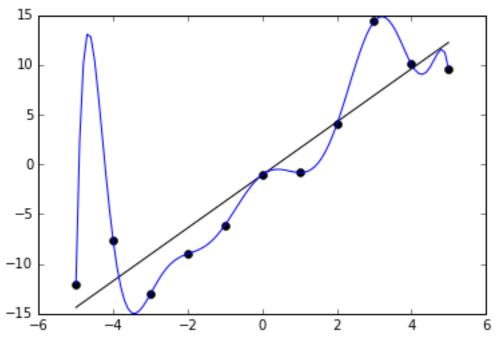


Figure 3.8: Noisy rougly linear data which has been fitted with a polynominal function.

which will let the agent assign each neuron to a spesific task to memorise the solution to the problem, instead of creating a generalization, and therefore be unable to accurately predict or classify new data being introduced.

Several methods exist to avoid overfitting, for example: cross-validation [32]; utilising different cost functions [18]; regularisation and weight decay [18, 23]; adding noise to the input [13, 23]; early stopping [13, 23]; and pruning, also called dropout [25].

In order to combat overfitting within my system I have introduced a gaussean noise to the input vectors which change with each epoch. They are only present in the very first layer of the network, and the amount of possible noise is a setting the user can decide. The noise for every input will be different for every epoch, and this will make the network more robust and resillient to overfitting.

I have also introduced a system where the network will test the training error and compare it to the best possible error previously experienced. If the error is lower, then the neurons will save the weights and deltas at that time. Then when the network is done training it will restore these weights. This is a variation on the early stopping method, with the difference being that the system keeps going until one of the other stopping conditions have been met.

Comparing the outputs

In order to simplify the network I have introduced a step function to round the outputs in my network:

$$\dot{y} = f(\sigma(t))) = \begin{cases} 0 \text{ for } \sigma(t) < 0.2 \\ 1 \text{ for } \sigma(t) > 0.8 \\ -1 \text{ for anything else} \end{cases} \tag{3.22}$$

This function is only used when comparing the target output with the actual output. If the output is later used as an input for a overlaying network, the output will be the pure $\sigma(t)$. 1 indicates activation, 0 indicates a non-responsive output and -1 represents an insufficiently tuned output. When comparing the results we will have three possible outcomes: success, which is when the rounded output and target output is the same; error, which is when the rounded output is opposite of the target output; and unknown, which is when the rounded output is given as -1.

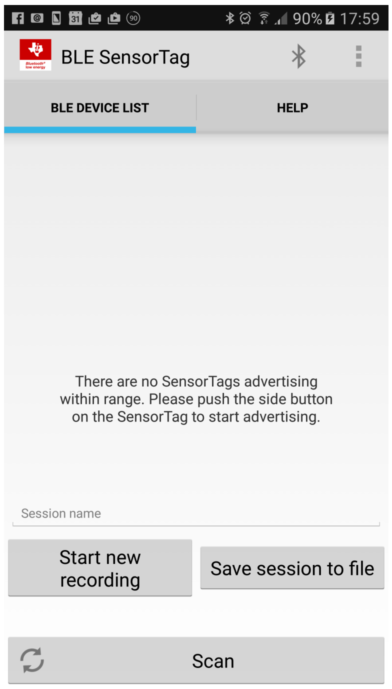
Chapter 4

Using the software

This chapter is a brief system walkthrough that shows how to use the system I have developed during the project, aswell as the settings used to obtain the final results presented in 5.4. There are two pieces of software developed: a modified version of the BLE Sensor Tag Android application, originally developed by Texas Instruments [8]; and the artificial neural network software which parses and processes the data collected by the Android application. Precompiled Windows binaries for the Neural Network are included in the attached CD/Flash Drive. *The modified BLE Sensor Tag application however does not have a compiled .apk, and needs to be installed through an IDE.*

4.1 The modified BLE Sensor Tag application

I found ten participants to perform the exercises described in 3.2, consisting of four women and six men. Table 4.1 shows a small exert from one of the experiments performed, excluding the timestamp which comes first, the surfixes described in Table 3.1 when there is data in the cell and rounded to four decimals. The experiment took place within the gymnasium hall and it's facilities just next to UiT.



There are no SensorTags advertising within range. Please push the side button on the SensorTag to start advertising.

Figure 4.1: BLE SensorTag screenshot

Figure 4.1 shows the main screen for this module. When wanting to record data we start by giving the session a name, which will be the file name aswell. In other words we will follow the naming convention described in 3.3.2 on the session name. Once a name has been selected we hit the "Start new recording" button in order to set it into recording mode. Following this we select the sensor we want to use from the list. This will take us to a new screen, which shows the activities of the various sensors: the movement sensor and barometric sensor.

To terminate the recording of data simply go back to the main screen, by hitting the back button on the android device, and hit the stop recording button. Then the data is prepared to be saved by using the save session to file. This will save the session to a file with the same name as the session with the added extension of the session's timestamp and extention .csv to the application's public folder.

These samples are saved locally on the android device and needs to be manually

Table 4.1: Exert from participant 2, exercise 2, take 3

Accel_X	Accel_Y	Accel_Z	Gyro_X	Gyro_Y	Gyro_Z	Mag_X	Mag_Y	Mag_Z	Bar
3.9258	-0.10254	-0.0088	-0.9141	0.7422	-0.8594	-103.5	14.1667	48.5CZ	0.0UK
0.0UK	0.0UK	0.0UK	0.0UK	0.0UK	0.0UK	0.0UK	0.0UK	0.0UK	102113.28
3.9326	-0.1133	-0.01270	-0.6875	0.6484	-0.5781	-106.1667	15.1667	51.8333	0.0UK

extracted to the desktop. In this case a simple USB connection between the android device and desktop sufficed.

The barometric pressure sensor

While not much to show on the data sets, there are two aspects of the data sets I would like to highlight. Figures 4.2-4.5 show the plots of one of the participant's walking up a set of stairs. As we can see in Figure 4.2 and 4.4 we can see clear patterns of movement, however the barometer, Figure 4.5, was far more difficult to see anything conclusive.

In this case we should see a gradual decrease in the atmospheric pressure, however this is not happening. In the beginning we can see it jumping up and down before stabilizing for several seconds, until it finally drops a little at the top of the stairs. This behaviour is however not consistent. Some times the barometric value stays around 102115 Pascal; in other samples it can sporadically jump between the values shown in the figure; and in some samples it flatlines.

The sensor as described from the manufacturers, see the paragraph about the sensor in Section 3.1.1, is capable of detecting shifts in height within 1 m margins, however the recieve during the experiments did not show this. It also did not show any changes when altitudes changed by higher amount.

It was tested by walking up stairs, however no noticable changes in the output were detected. After discussing it with peers, we speculate this discrepancy could be caused by the environment of which the experiments were taking place in. As the sensor is dependant on detecting air pressure, ventilation systems could

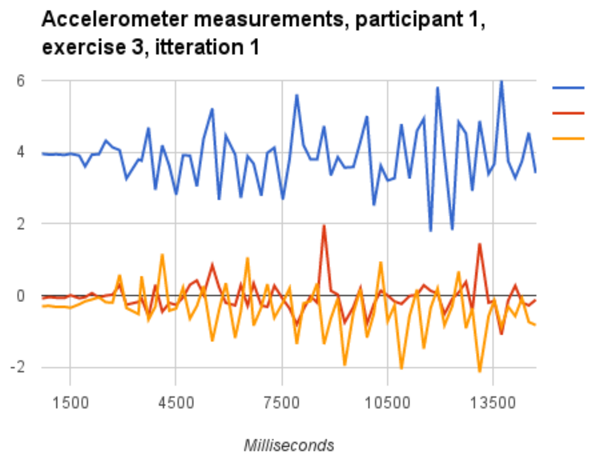


Figure 4.2: Plot of accelerometer data from the first participant's first execution of exercise 3. Blue - x, red - y, yellow, z

influence the data. In order to challenge this, I also attempted to test the stairs in various places on campus, however no significant differences in the output were observed.

It could also be that the default settings for the application are not made to be as sensitive as I need it. According to the system documents for the BMP280 it should be sensitive enough to detect variations within ± 0.12 hPa, or ± 1 meter difference in altitude, hence it should easily be possible to see the differences in pressure when taking the device up and down stairs.

Lastly it could be that the sensor's filter settings are not configured correctly. Reading through the datasheets about the sensor on their page shows several use cases with different settings [30, Table 7]. It could simply be that the sensor's filter settings are wrong for this use case.

Regardless the reason, because of the unreliable data I elected to exclude the barometric pressure sensor from the data sets and network, hence the CSVReader ignores the lines which contains barometric data when it parses files.

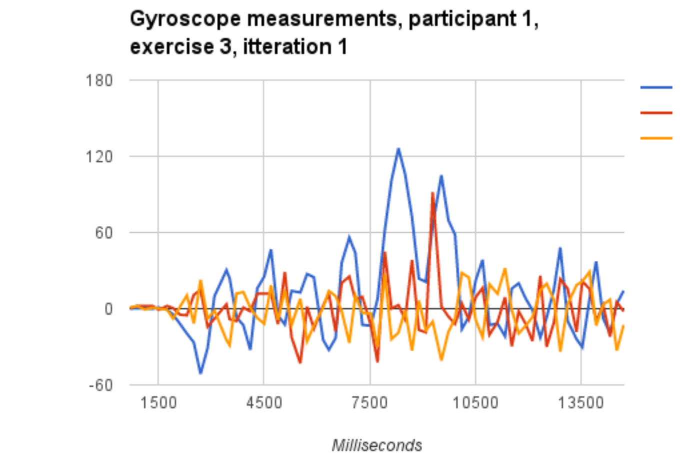


Figure 4.3: Plot of gyroscope data from the first participant's first execution of exercise 3. Blue - x, red - y, yellow, z

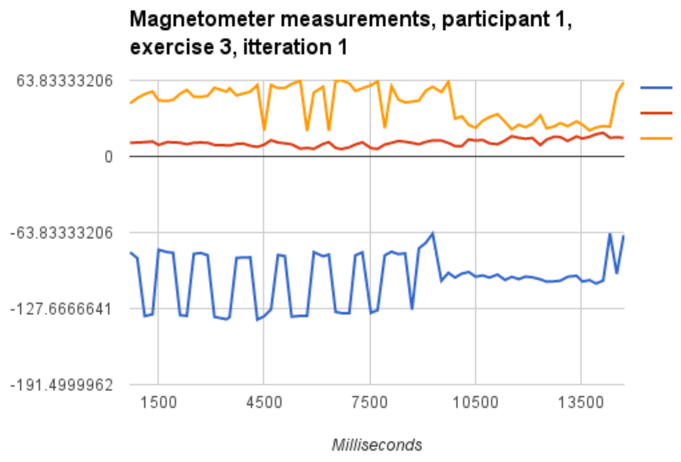


Figure 4.4: Plot of magnetometer data from the first participant's first execution of exercise 3. Blue - x, red - y, yellow, z

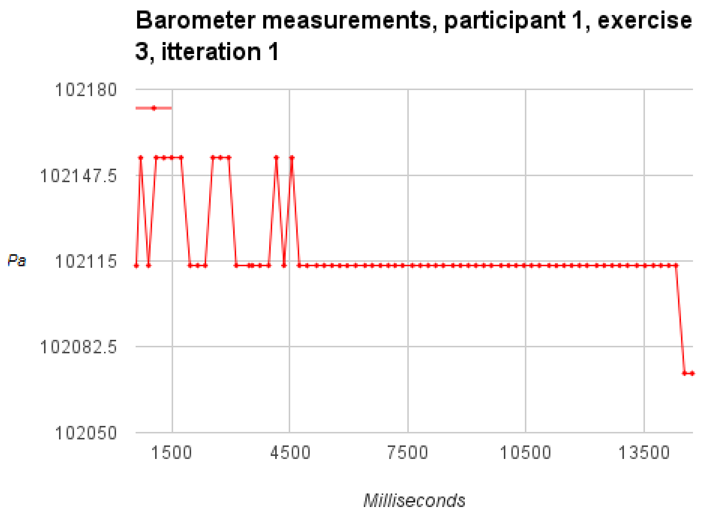


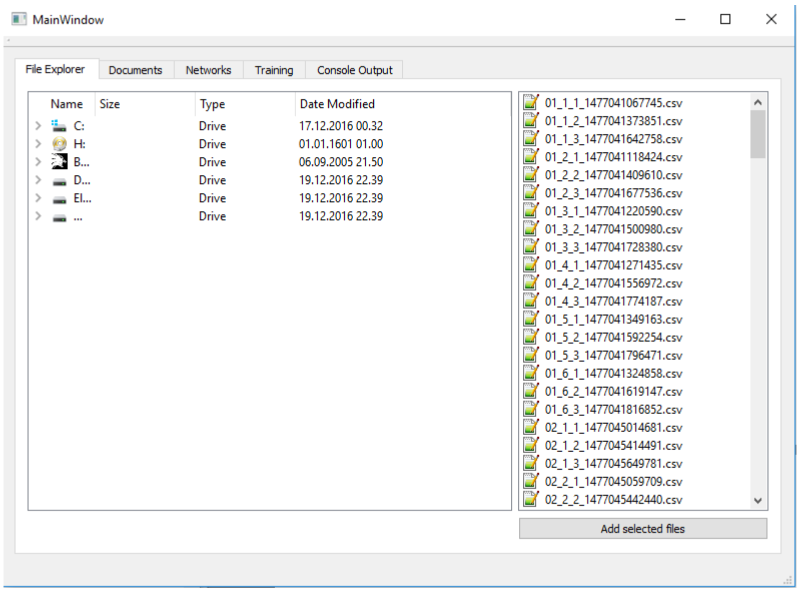
Figure 4.5: Plot of barometer data from the first participant's first execution of exercise 3.

4.2 Back Propagation Network

In order to analyse the data given by the BLE Sensor Tag application I have developed a rudimentary program which is programmed spesifically for analysing it's data. While the software could easily be modified to accomodate any sort of data, the structure of the program's document parser and the way the network extracts the data for processing is specified for analysing the data given by the BLE Sensor Tag.

4.2.1 File explorer

The very first step of using the software is to select the files we're going to analyse. I have included a default data set, which I used to train the network, but if one wants to use other data this is where it's done. Find the files with the explorer, as shown in Figure 4.6, and simply add the files to the list of eligible files. There is



Name	Size	Type	Date Modified
C:		Drive	17.12.2016 00.32
H:		Drive	01.01.1601 01.00
B...		Drive	06.09.2005 21.50
D...		Drive	19.12.2016 22.39
El...		Drive	19.12.2016 22.39
...		Drive	19.12.2016 22.39

01_1_1_1477041067745.csv
01_1_2_1477041373851.csv
01_1_3_1477041642758.csv
01_2_1_1477041118424.csv
01_2_2_1477041409610.csv
01_2_3_1477041677536.csv
01_3_1_1477041220590.csv
01_3_2_1477041500980.csv
01_3_3_1477041728380.csv
01_4_1_1477041271435.csv
01_4_2_1477041556972.csv
01_4_3_1477041774187.csv
01_5_1_1477041349163.csv
01_5_2_1477041592254.csv
01_5_3_1477041796471.csv
01_6_1_1477041324858.csv
01_6_2_1477041619147.csv
01_6_3_1477041816852.csv
02_1_1_1477045014681.csv
02_1_2_1477045414491.csv
02_1_3_1477045649781.csv
02_2_1_1477045059709.csv
02_2_2_1477045442440.csv

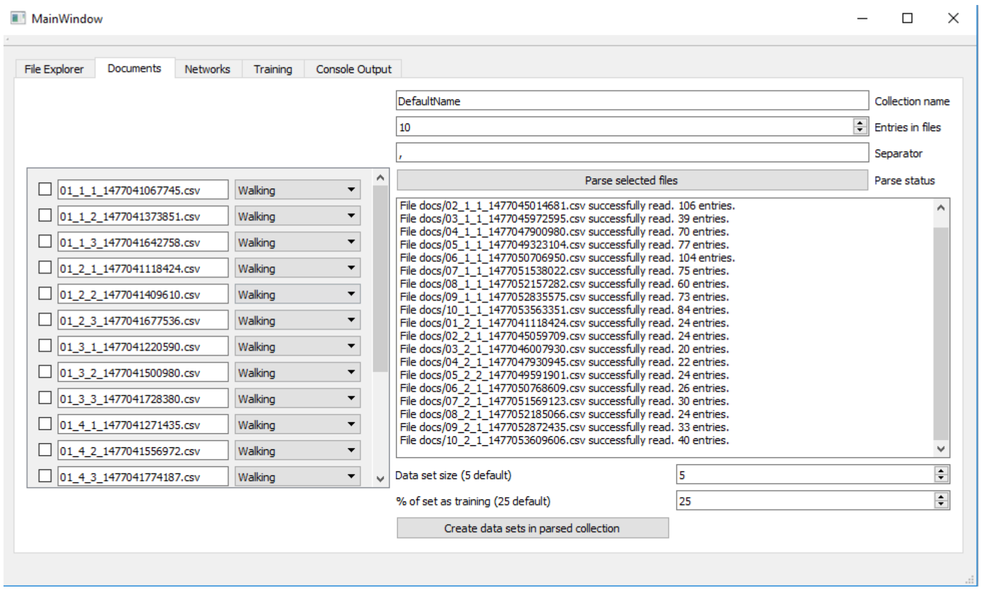
Figure 4.6: Main Window File Explorer tab

a security net which prevents the user from adding files apart from .csv files, but this does not inhibit the user from adding files which do not have the structure needed to be read by the system.

4.2.2 Document parser

Next in order to parse the files we need to identify what data they contain. The comboboxes on the left, in Figure 4.7 lets the user identify the content of the file, and which activity is being performed in it. Currently the system does not support files filled with data to identify what activity is going on. Furthermore the user needs to check which files are to be parsed to a collection, in case the user wants to separate the files into different sets. It is however important to note that a single collection needs to contain sufficient falling and non-falling activities in order to train the system properly.

The user gives the collection a name, which is how we identify the collection



Collection name	DefaultName
Entries in files	10
Separator	,
Data set size (5 default)	5
% of set as training (25 default)	25

☐ 01_1_1_1477041067745.csv	Walking
☐ 01_1_2_1477041373851.csv	Walking
☐ 01_1_3_1477041642758.csv	Walking
☐ 01_2_1_1477041118424.csv	Walking
☐ 01_2_2_1477041409610.csv	Walking
☐ 01_2_3_1477041677536.csv	Walking
☐ 01_3_1_1477041220590.csv	Walking
☐ 01_3_2_1477041500980.csv	Walking
☐ 01_3_3_1477041728380.csv	Walking
☐ 01_4_1_1477041271435.csv	Walking
☐ 01_4_2_1477041556972.csv	Walking
☐ 01_4_3_1477041774187.csv	Walking

Parse status

```
File docs/02_1_1_1477045014681.csv successfully read. 106 entries.
File docs/03_1_1_1477045972595.csv successfully read. 39 entries.
File docs/04_1_1_1477047900980.csv successfully read. 70 entries.
File docs/05_1_1_1477049323104.csv successfully read. 77 entries.
File docs/06_1_1_1477050706950.csv successfully read. 104 entries.
File docs/07_1_1_1477051538022.csv successfully read. 75 entries.
File docs/08_1_1_1477052157282.csv successfully read. 60 entries.
File docs/09_1_1_1477052835575.csv successfully read. 73 entries.
File docs/10_1_1_1477053563351.csv successfully read. 84 entries.
File docs/01_2_1_1477041118424.csv successfully read. 24 entries.
File docs/02_2_1_1477045059709.csv successfully read. 24 entries.
File docs/03_2_1_1477046007930.csv successfully read. 20 entries.
File docs/04_2_1_1477047930945.csv successfully read. 22 entries.
File docs/05_2_2_1477049591901.csv successfully read. 24 entries.
File docs/06_2_1_1477050768609.csv successfully read. 26 entries.
File docs/07_2_1_1477051569123.csv successfully read. 30 entries.
File docs/08_2_1_1477052185066.csv successfully read. 24 entries.
File docs/09_2_1_1477052872435.csv successfully read. 33 entries.
File docs/10_2_1_1477053609606.csv successfully read. 40 entries.
```

Figure 4.7: Document parser and data set tab

later, and determine the number of entries in the files and separator. We do not need to change anything here, as the presets are set for the experimental data obtained in 4.1. Then the files are parsed into a Data Collection.

We then need to determine the training- and testing-sets' parameters. One part of these sets are called Data Segments, and they're built by determining an n size, or length, to the set, default of which is 5. We thus create:

$$i_{max} = number\_of\_entries\_in\_collection - n \tag{4.1}$$

data segments from the parsed collection of documents.

The size of the Data Segment determines the number of inputs we give to the neural network, and thus the number of input nodes the network needs. This is equal to three times the size of the segment plus one. With the default size of 5 we get a data segment which yields 16 inputs for each sensor: 1 input, $||a_i||$ which is the total length of the values within the segment:

$$||a_i|| = \sqrt{\sum_{j=0}^{n-1} x_{i+j}^2 + \sum_{j=0}^{n-1} y_{i+j}^2 + \sum_{j=0}^{n-1} z_{i+j}^2} \tag{4.2}$$

and the 5 values for x, y and z.

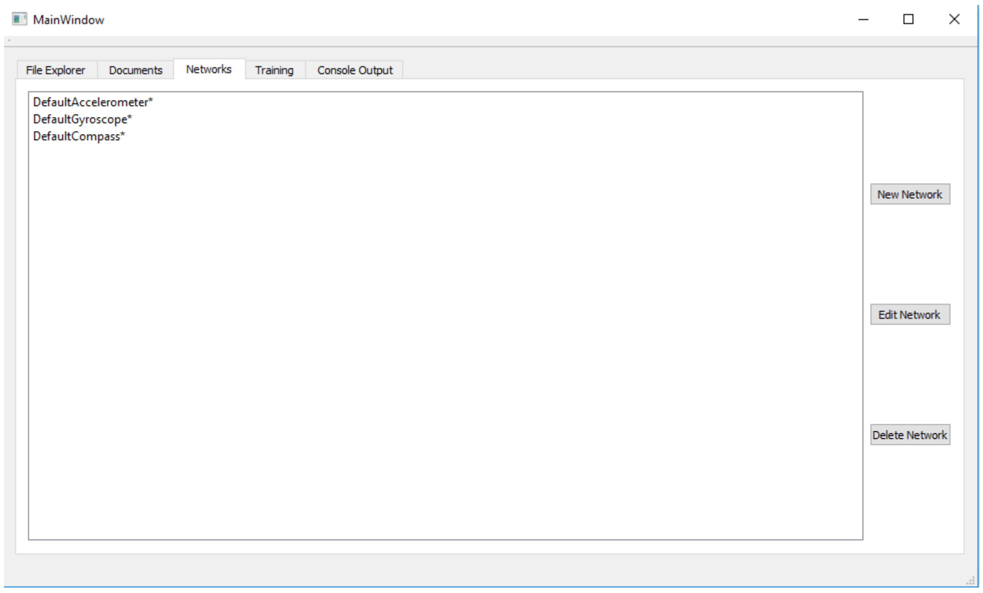
Lastly we need to decide how large part of the entire collection should be used for training, and by extension how many should be used for testing. The default value is currently set to 25%, however I believe smaller sets may be more appropriate. Because of this the default sets I have used to train the network only has 15% of the segments used for training. Once the training set size is determined the collection of segments are randomly shuffled and divided into training and testing sets.

In order to simplify the processes of training and testing I have included three default data collections: "First training run", which contains the data used in the networks labled with "First" and contains only data from exercise 1 and 2, see 3.2; "Refined network", which contains the data used to train the networks labeld "Refined" and it contains all six exercises, however only 3 files per exercise; and "All experiments", which contains all the data collected during the experimentation and this set has been used to test the networks once training is complete.

4.2.3 Neural Network creation

This tab is responsible for managing the networks and their structure. It contains a list of the networks, aswell as buttons for creating new, editing or deleting networks, see Figure 4.8.

There are two parts of the network creation dialog: creating a brand new simple neural network, see Figure 4.9; and creating a network with other networks as inputs, see Figure 4.10. Theoretically the system supports any depth with these networks, but I have only tested for with a depth of two.



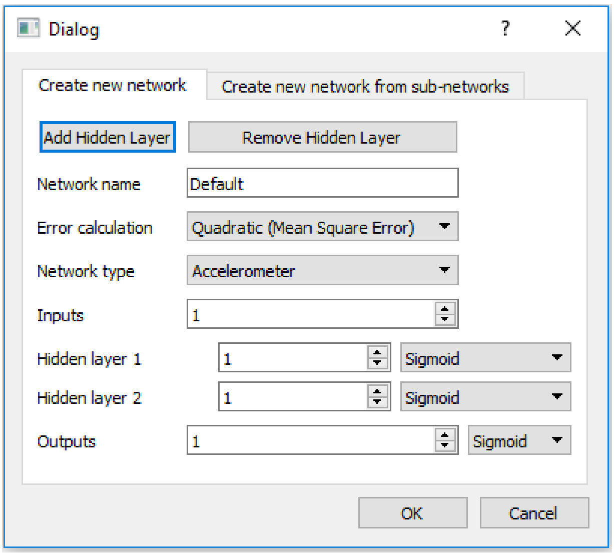
DefaultAccelerometer*
DefaultGyroscope*
DefaultCompass*

Figure 4.8: Network creation tab

Through the dialog the user can specify the network's name, which is how we identify the network although these names do not need to be unique as they have a hidden unique identity; the network's error, or cost, calculation although only Mean Square Error is currently supported; the network's type, which specifies what sensor data we are extracting from the training set; and the number of neurons to have in each layer aswell as their output type, though only sigmoid output is currently implemented. Finally in the "Create new network from sub-networks" the number of inputs is replaced by a list of existing networks which the network can use as inputs.

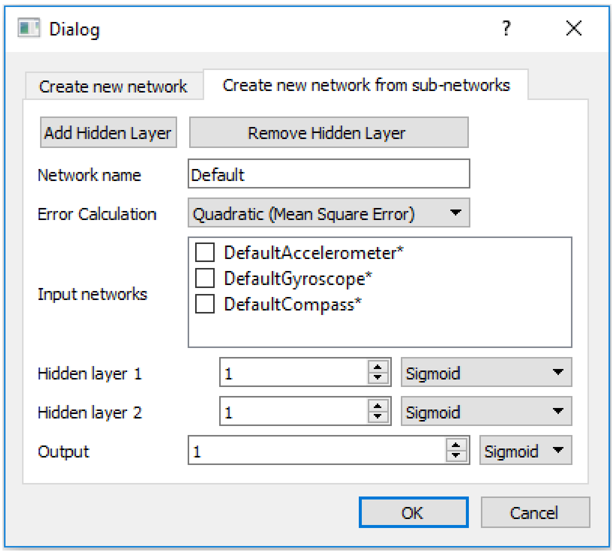
Should the user wish to edit a network this will display the Network creation dialog, but with the editorial mode instead, where they can edit any part of the network's structure. It is however currently not possible to edit networks which takes other networks as inputs, so if the user wants to edit such a network they will currently need to delete the old one and create a new.

Just like the document parser, to make training and testing more efficient,



Network name: Default
Error calculation: Quadratic (Mean Square Error)
Network type: Accelerometer
Inputs: 1
Hidden layer 1: 1 Sigmoid
Hidden layer 2: 1 Sigmoid
Outputs: 1 Sigmoid

Figure 4.9: Network creation dialog



Network name: Default
Error Calculation: Quadratic (Mean Square Error)
Input networks:
- ☐ DefaultAccelerometer*
- ☐ DefaultGyroscope*
- ☐ DefaultCompass*

Hidden layer 1: 1 Sigmoid
Hidden layer 2: 1 Sigmoid
Output: 1 Sigmoid

Figure 4.10: Network creation by using subnetworks dialog

there exists a couple of presets networks. With the final build of this software these are the settings used to get the results in: 5.1.2, which are the networks labled “First”; and 5.1.3, which are labled “Refined”. I will not go into detail for the “First” networks here, see 5.1.2 for those details, but the structure of the “Refined” networks are as follows:

Structure: Network layer 1, network 1 - Accelerometer network

- Inputs: 16
- Hidden: 16, 6
- Outputs: 2

Structure: Network layer 1, network 2 - Gyroscope network

- Inputs: 16
- Hidden: 16, 6
- Outputs: 2

Structure: Network layer 1, network 3 - Magnetometer network

- Inputs: 16
- Hidden: 2, 3
- Outputs: 2

Structure: Network layer 2, network 1 - Top network

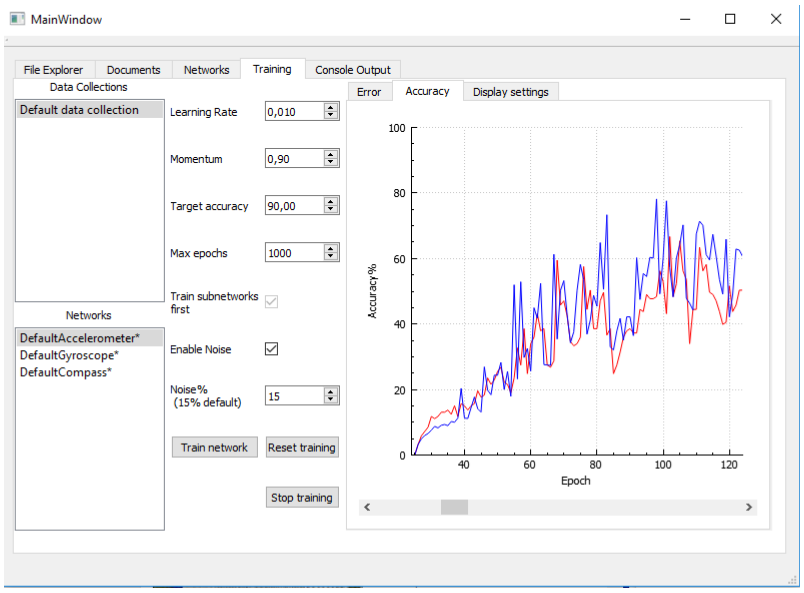
Unlike the other networks this network has a preset amount of input nodes, no matter the size of the size of the data segments used in the underlying networks: six, two for each of the underlying networks' outputs. Therefore this network has the following structure:

- Inputs: Accelerometer network, Gyroscope network, Magnetometer network
- Hidden: 2, 4
- Outputs: 2

4.2.4 Neural Network training

The final figure, Figure 4.11, shows the training window which is where we select which networks to train, which data sets to use, and the parameters for the training. In addition it also includes graphs to illustrate the accuracy and error per epoch for the networks, aswell as settings for how these are rendered. Training a network will generate a file which contains the accuracy and error for the testing and training sets within the error, and the epoch with the lowest error will be noted in the file's name. It is worth noting that it's not necessary to train the underlying networks, one can assign the top network for training and it'll train the underlying networks automatically with the learning parameters set for the top network. This will however clutter the graphs with all the networks at once, so it changing the display settings might be necessary during runtime of the network. Should one want to prematurely stop the training of a subnetwork during this, one needs to select that particular subnetwork in the list and click the stop training button.

Should one however want to exercise more control over the learning of each subnetwork it is possible to set their settings and train them individually. We will



Learning Rate 0,010
Momentum 0,90
Target accuracy 90,00
Max epochs 1000
Train subnetworks first ☐
Enable Noise ☑
Noise% (15% default) 15

Figure 4.11: Network training tab

see more about the effects of this in 5.2.2.

Finally there is a button missing from the figure. This is the "Test network" button, which lets the user run the entire selected data set through a network and get an analysis of the performance on the entire set, like a testing set epoch, but with much greater details. This will create a .csv file named after the network and data set containing the accuracy of the set, the set's error, aswell as a compilation of the network outputs compared to the targets for every set.

Chapter 5

Results and discussion

5.1 Results

5.1.1 Overview

Table 5.1: Overview of the tests with their training data, networks and learning parameters

Test number	Data set	Networks	Learning Rate	Momentum	Target Accuracy	Max epochs
1	First training run	FirstAccelerometer	0,01	0,90	80	1000
		FirstGyroscope	0,01	0,90	80	1000
		FirstCompass	0,01	0,90	80	1000
		FirstTopNetwork	0,01	0,90	80	1000
2	Refined Network	RefinedAccelerometer	0,0025	0,90	90	5000
		RefinedGyroscope	0,0025	0,90	90	5000
		RefinedCompass	0,0025	0,90	90	5000
3	Refined Network	RefinedAccelerometer	0,0005	0,90	80	1000
		RefinedGyroscope	0,0005	0,90	80	1000
		RefinedCompass	0,0005	0,90	80	1000
		RefinedTopNetwork	0,0005	0,90	80	1000
4	Refined Network	RefinedAccelerometer	0,0025	0,90	90	5000
		RefinedGyroscope	0,0025	0,90	90	5000
		RefinedCompass	0,0005	0,90	80	5000
		RefinedTopNetwork	0,0001	0,90	80	5000

Table 5.2: Description of what the tests entail

Test number	Description
1	First run of a complete neural network.
2	Finding the structure of subnetworks.
3	Finding the structure of the top network and adjusting parameters.
4	Finalise learning parameters for each network.

From *Test 2* and onwards a test of the network was performed with the “All experiments” data set in order to validate the network on unknown data. These network tests will be the basis of the results.

Table 5.3: Structures of the networks

Network name	Input neurons	Hidden neurons	Output neurons
FirstAccelerometer	16	16 — 6	2
FirstGyroscope	16	16 — 6	2
FirstCompass	16	16 — 6	2
FirstTopNetwork	Accel — Gyro — Compass	16 — 6	2
RefinedAccelerometer	16	16 — 6	2
RefinedGyroscope	16	16 — 6	2
RefinedCompass	16	2 — 3	2
RefinedTopNetwork	Accel — Gyro — Compass	2 — 4	2

5.1.2 Test 1 - First test run of a complete network

The first test was made with the neural network in order to check if the network would properly feed forward and backpropagate.

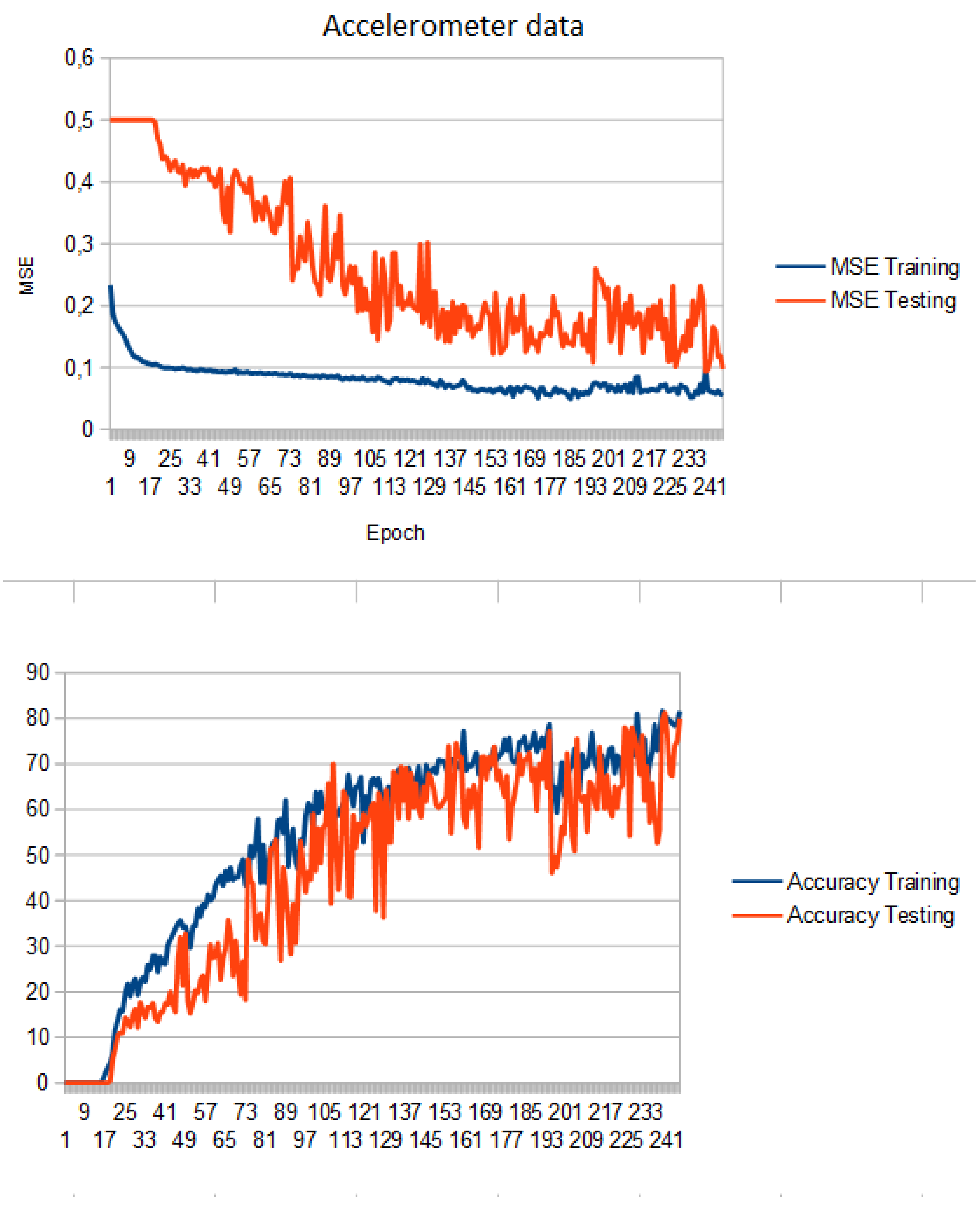
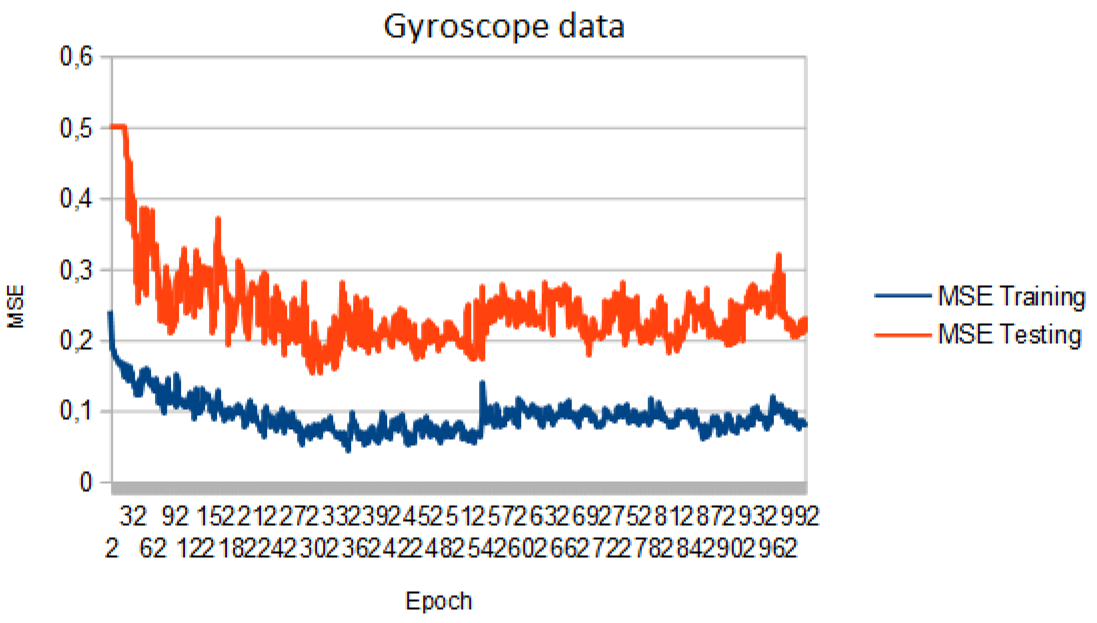


Figure 5.1: First full accelerometer MSE and Accuracy, Test 1



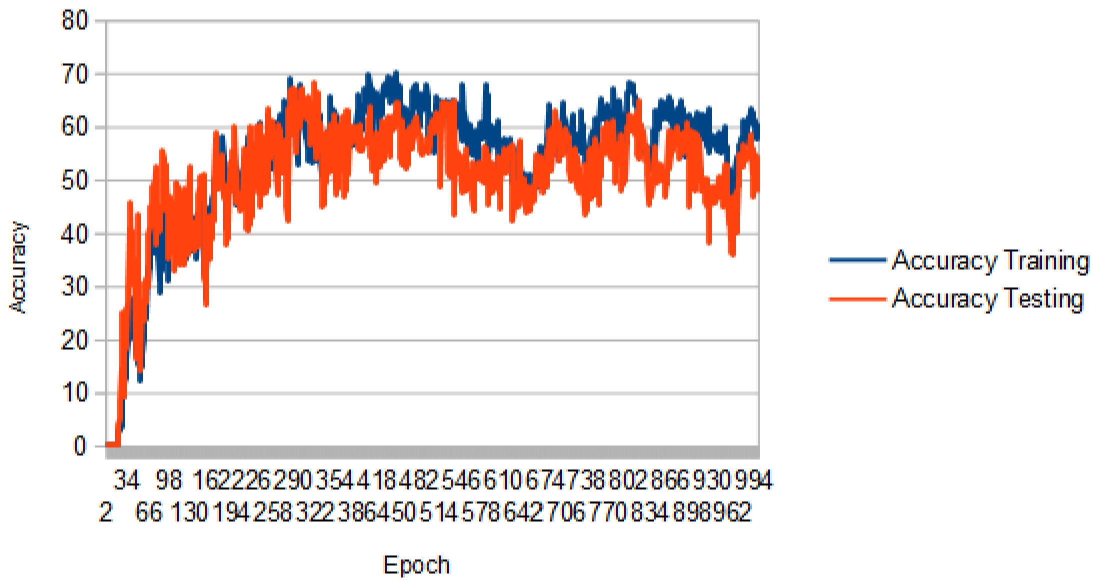
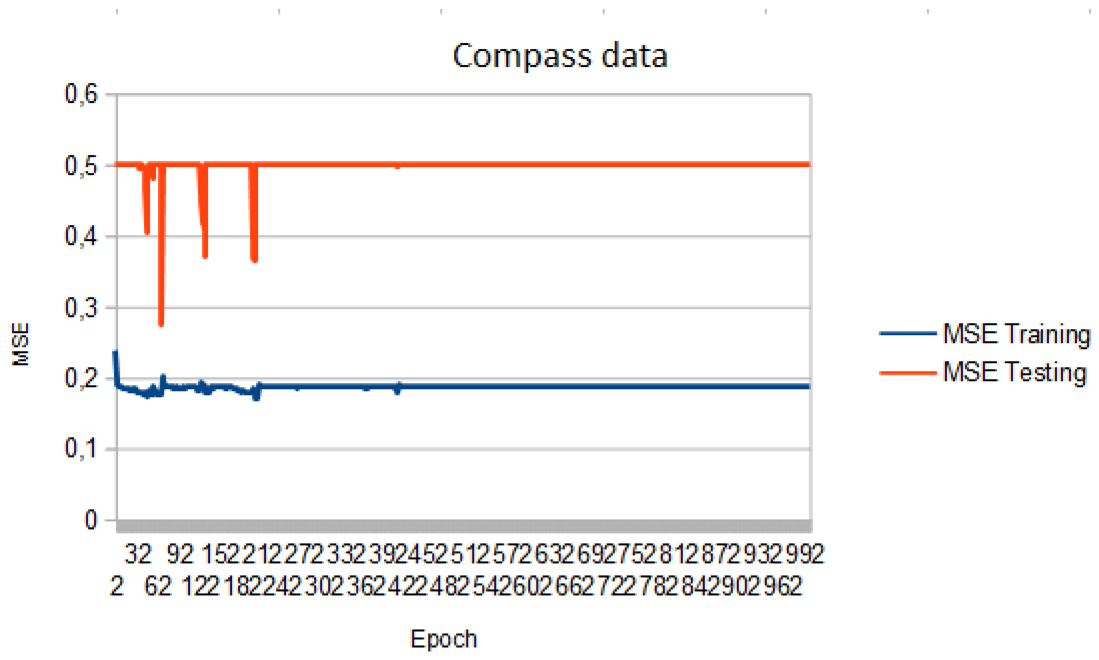


Figure 5.2: First full gyroscope MSE and Accuracy, Test 1



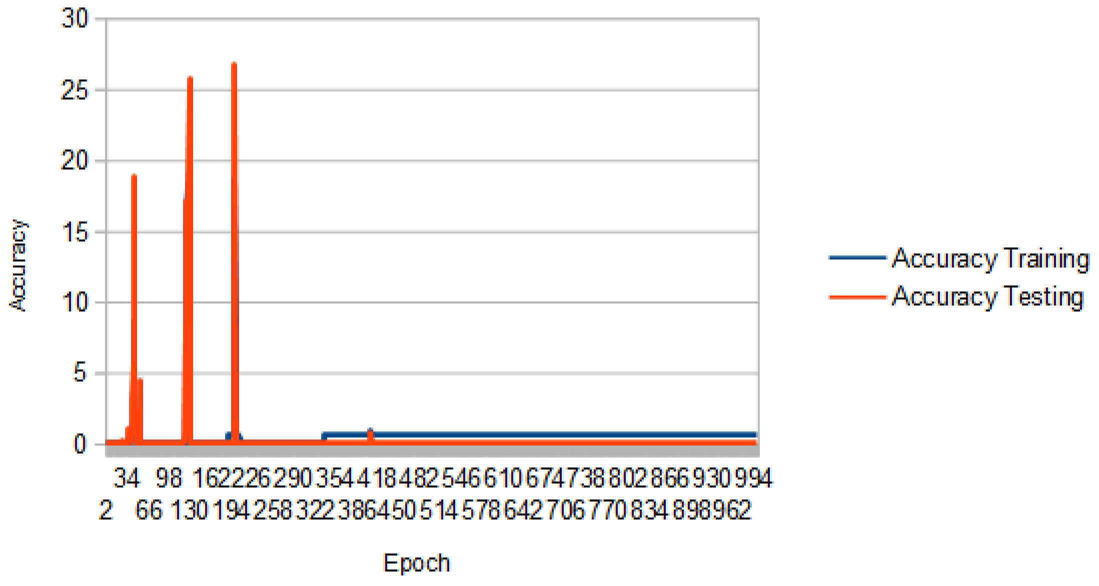
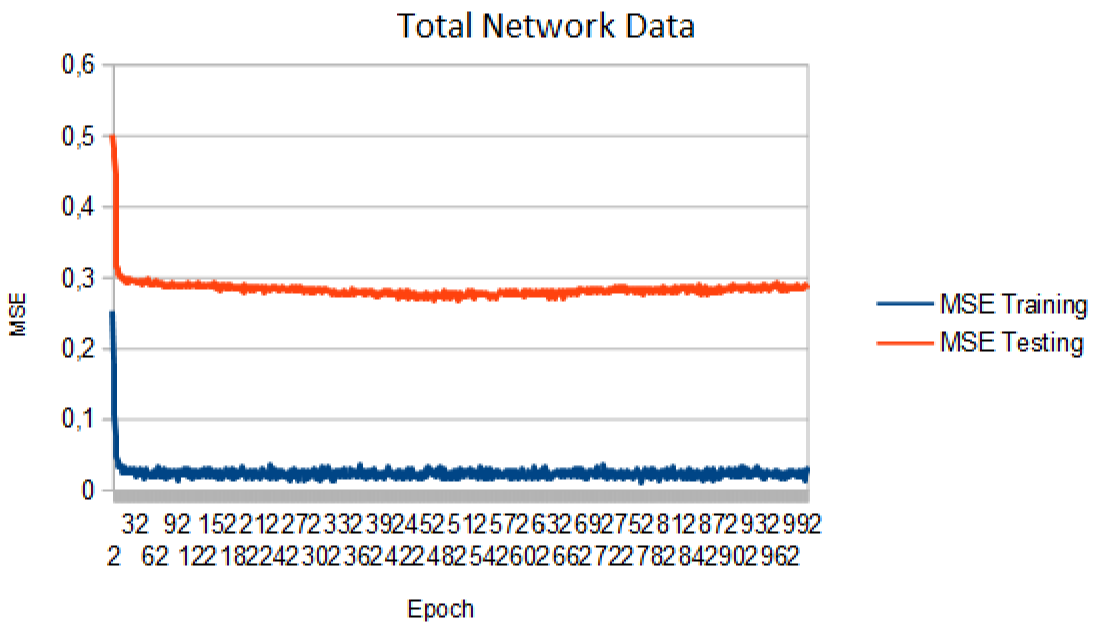


Figure 5.3: First full compass MSE and Accuracy, Test 1



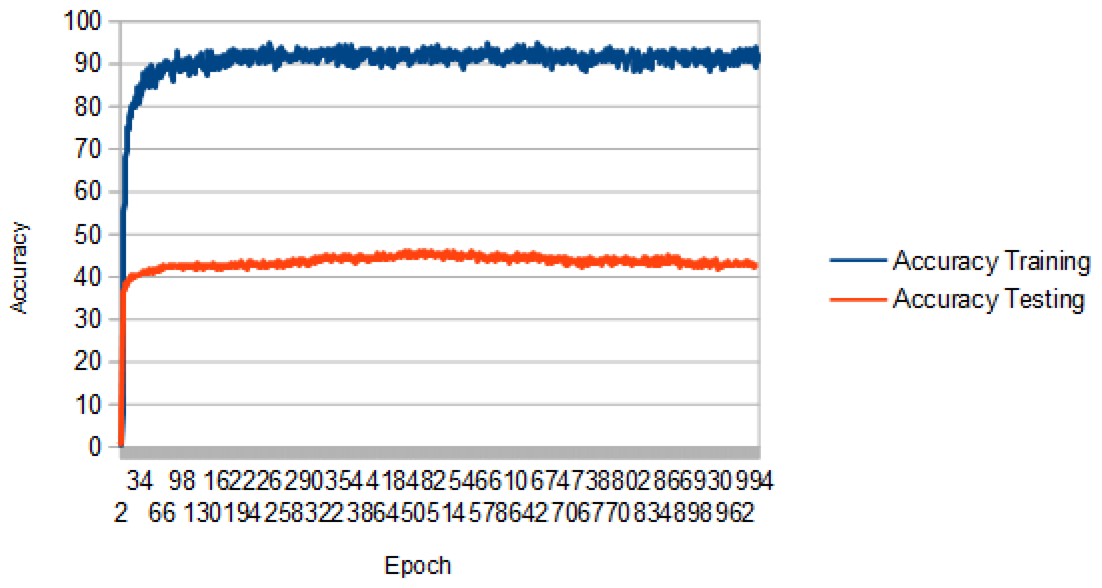


Figure 5.4: First full total network MSE and Accuracy, Test 1

5.1.3 Test 2-4 - Refining the network

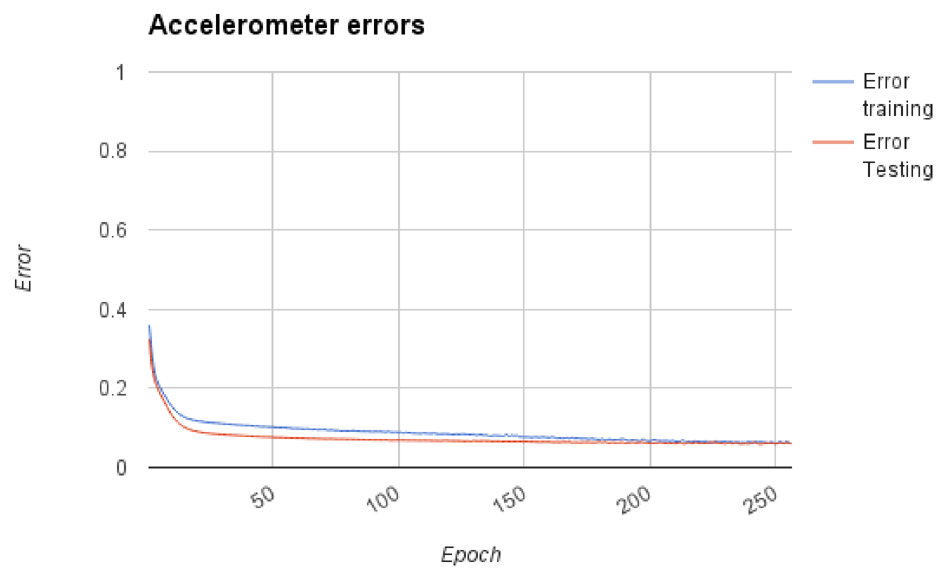


Figure 5.5: Test 2 accelerometer error

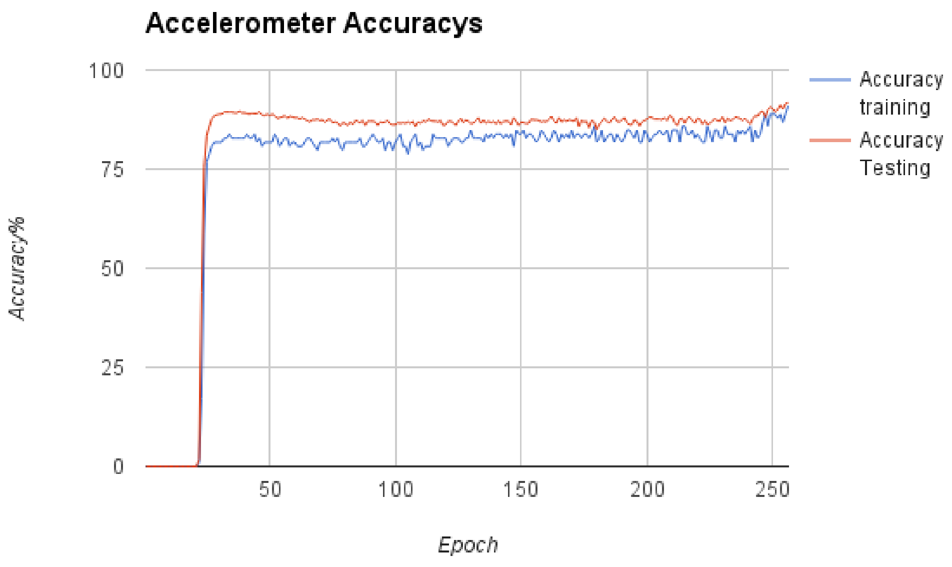


Figure 5.6: Test 2 accelerometer accuracy

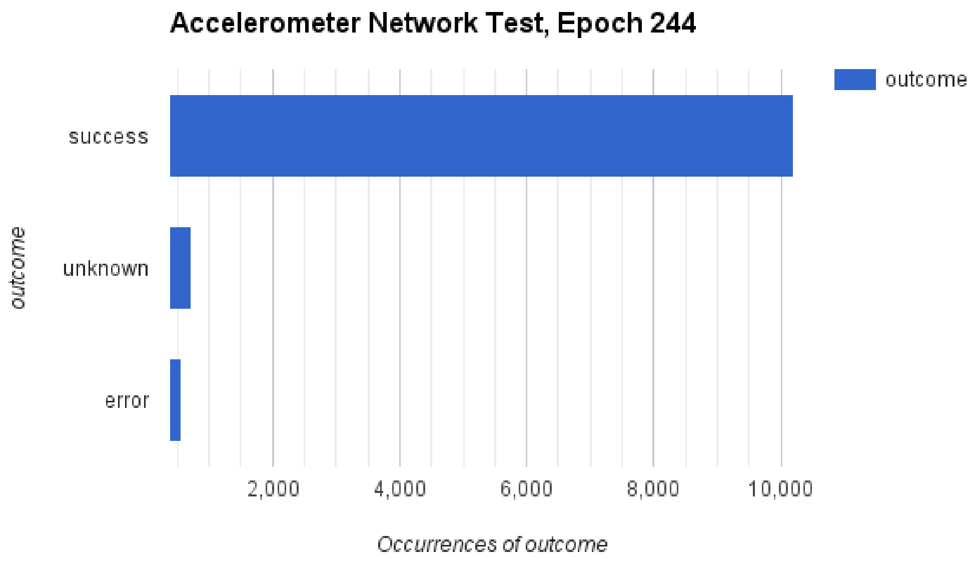


Figure 5.7: Test 2 accelerometer network test

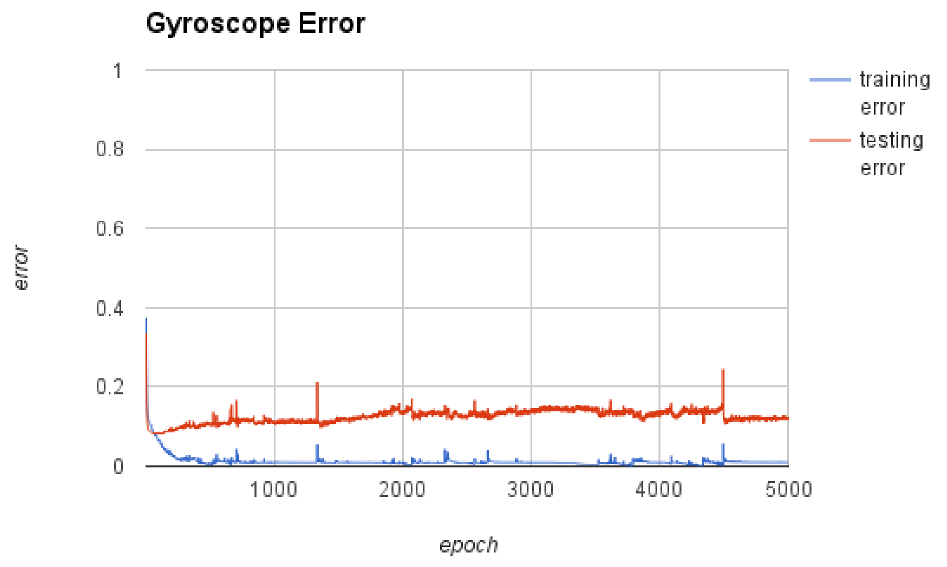


Figure 5.8: Test 2 gyroscope error

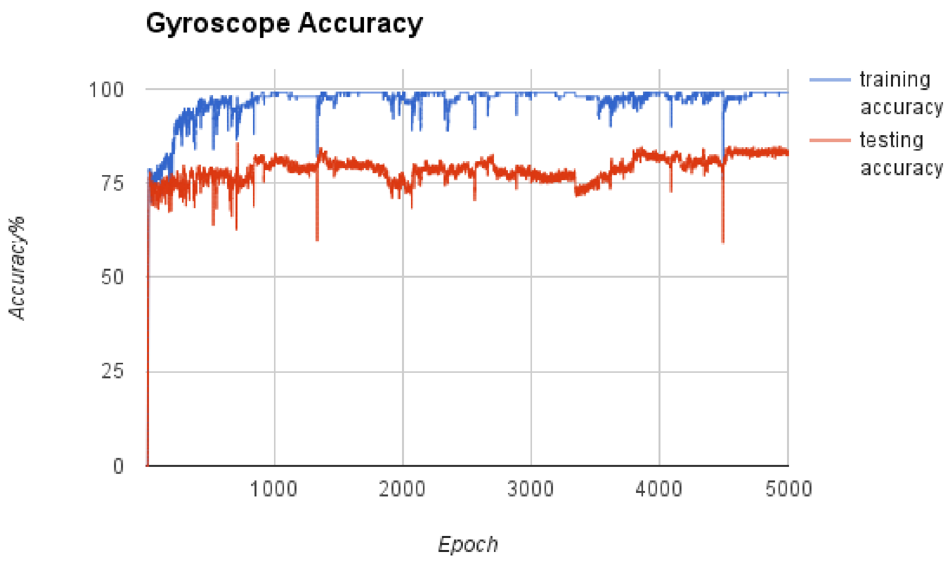


Figure 5.9: Test 2 gyroscope accuracy

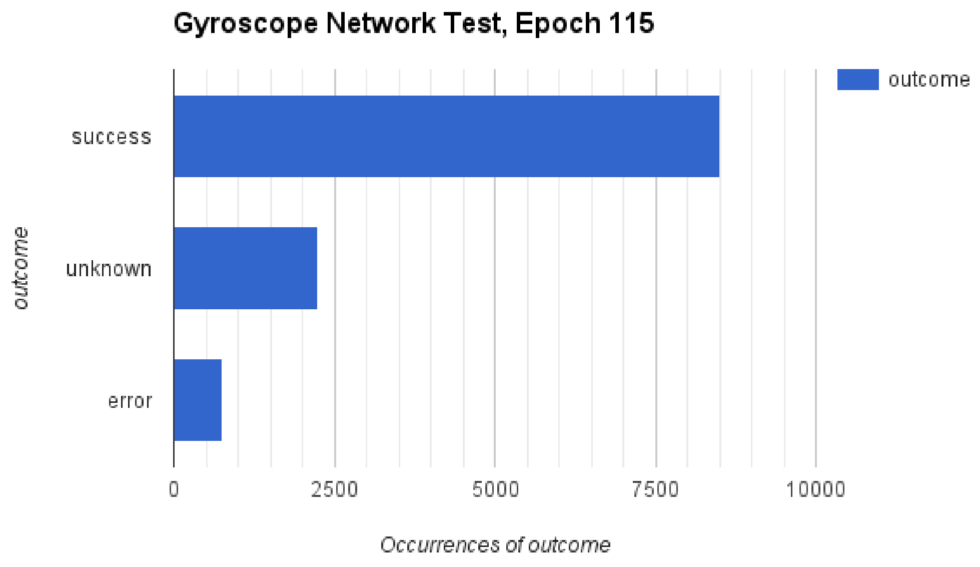


Figure 5.10: Test 2 gyroscope network test

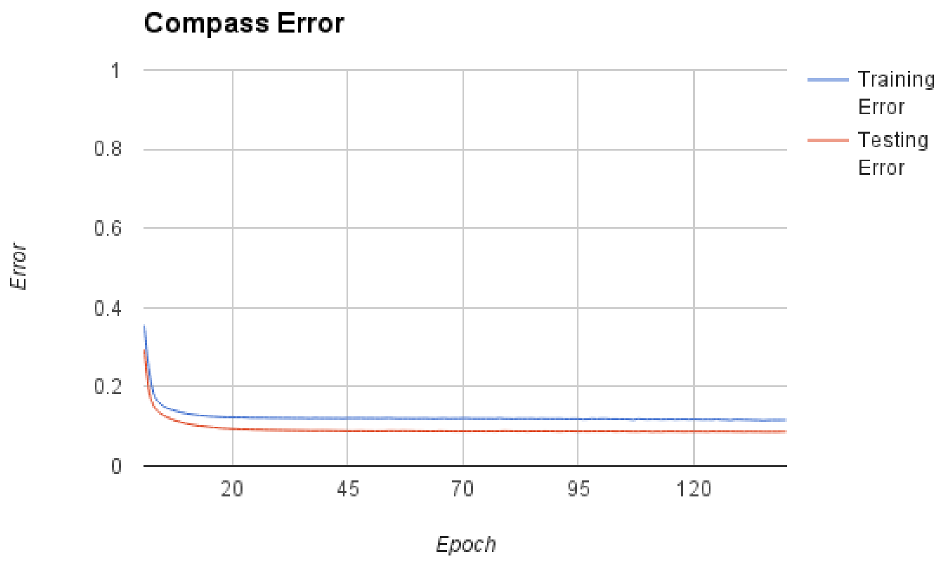


Figure 5.11: Test 2 magnetometer error

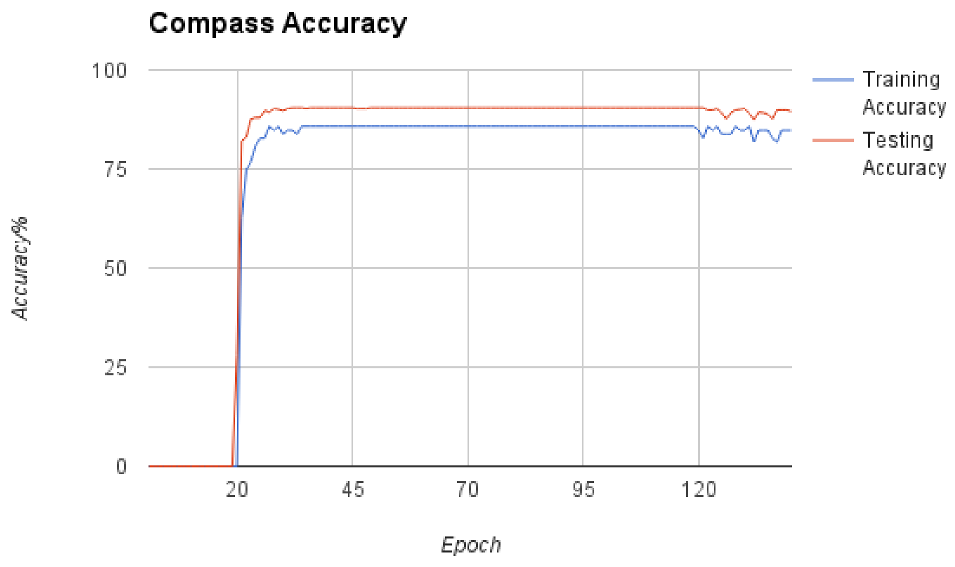


Figure 5.12: Test 2 magnetometer accuracy

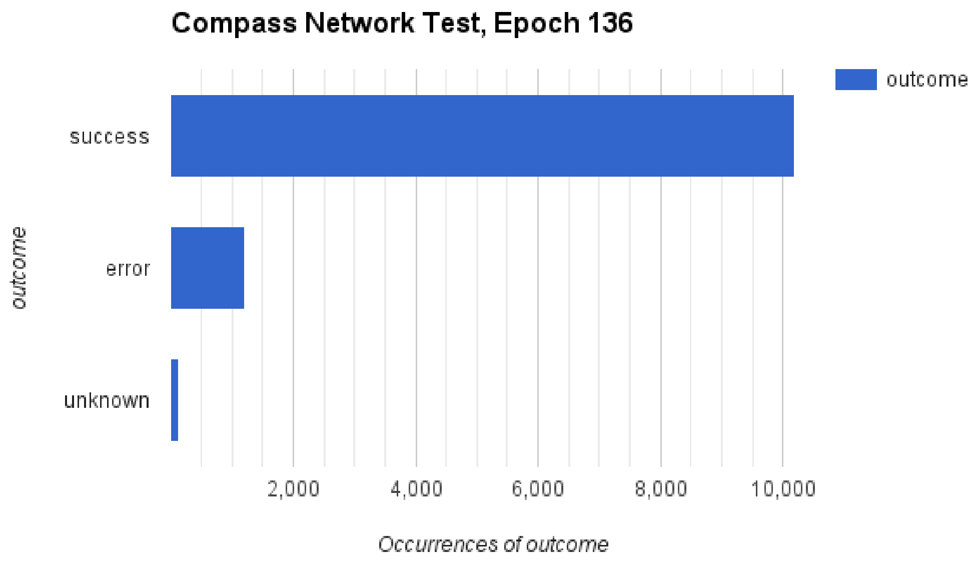


Figure 5.13: Test 2 magnetometer network test

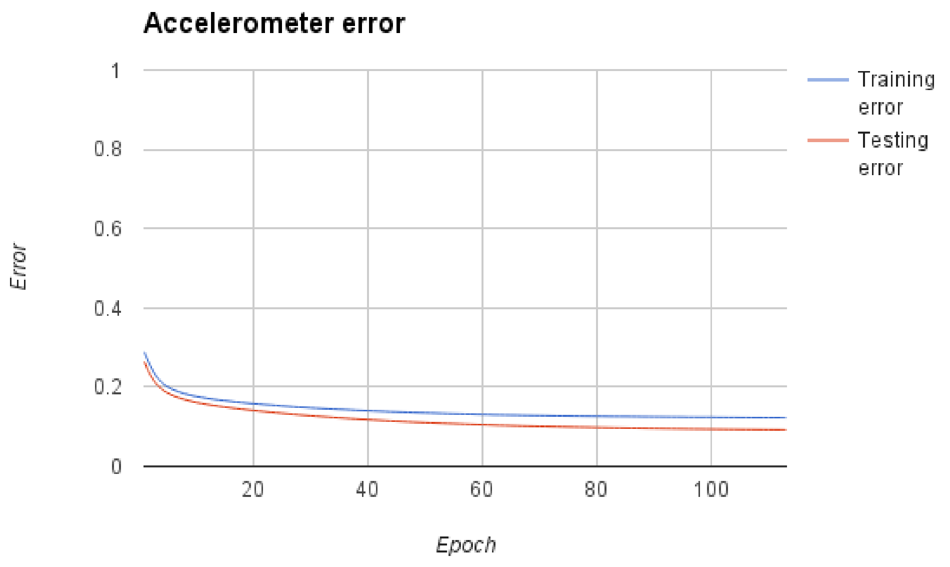


Figure 5.14: Test 3 accelerometer error

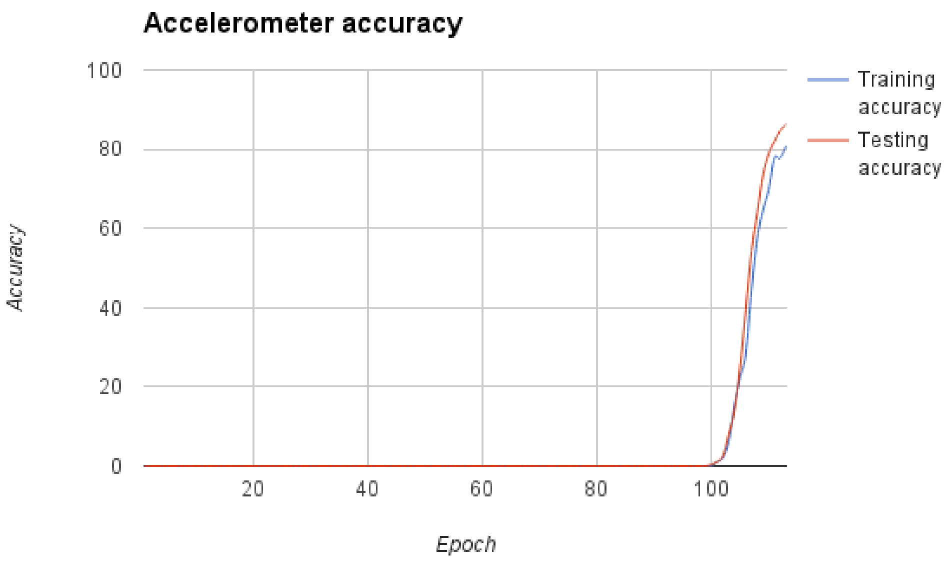


Figure 5.15: Test 3 accelerometer accuracy

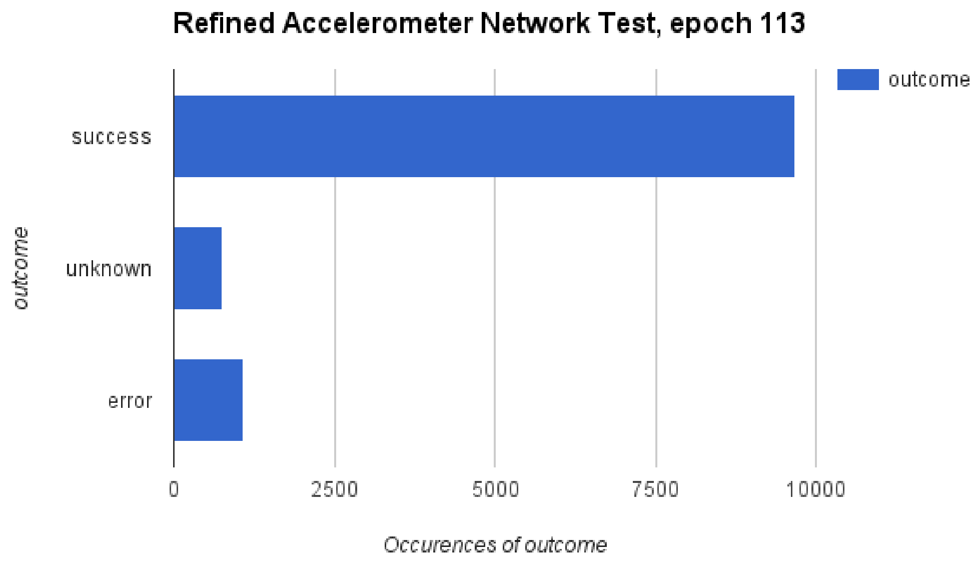


Figure 5.16: Test 3 accelerometer network test

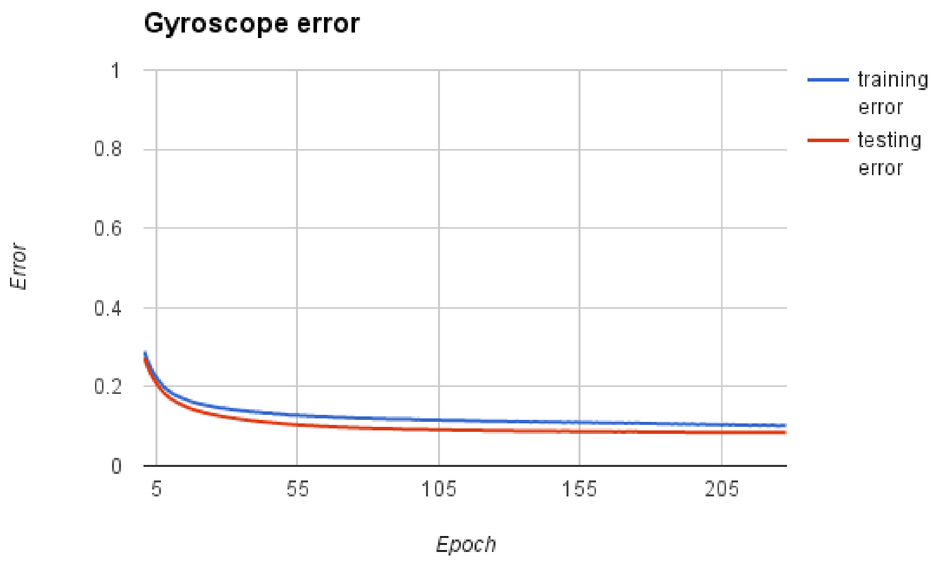


Figure 5.17: Test 3 gyroscope error

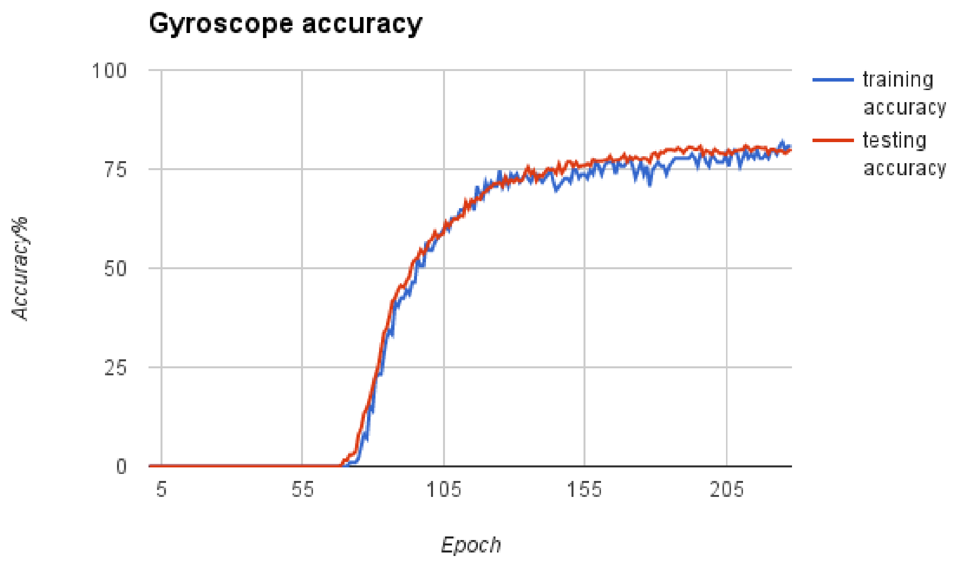


Figure 5.18: Test 3 gyroscope accuracy

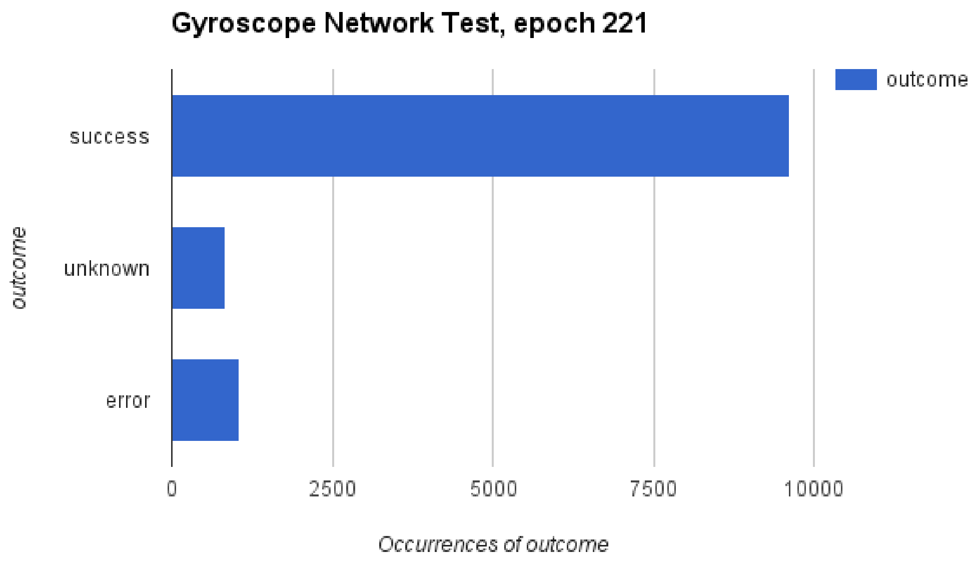


Figure 5.19: Test 3 gyroscope network test

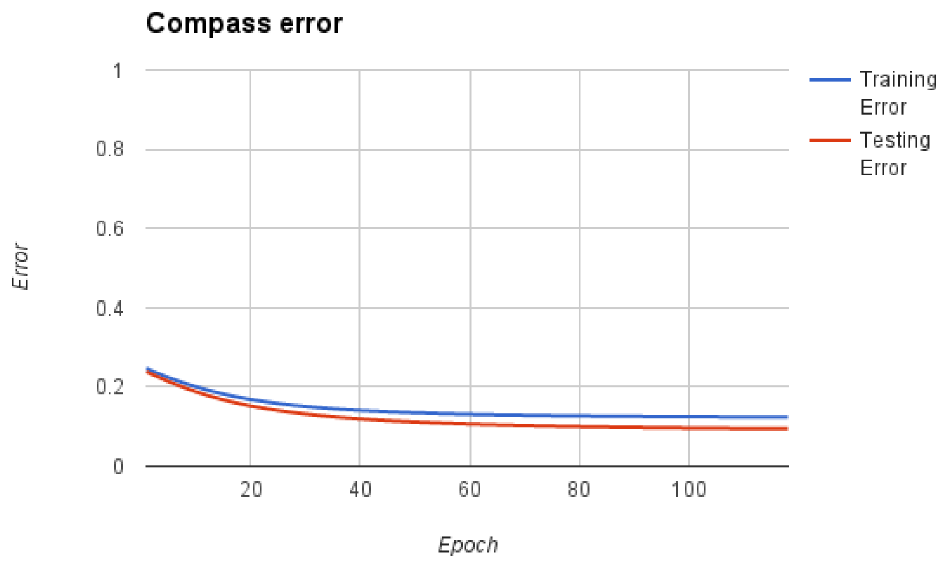


Figure 5.20: Test 3 magnetometer error

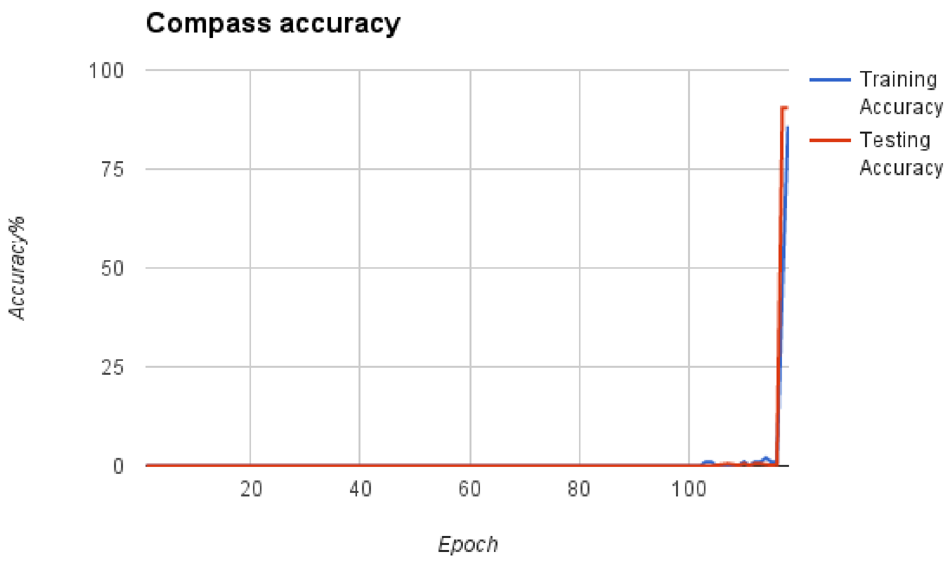


Figure 5.21: Test 3 magnetometer accuracy

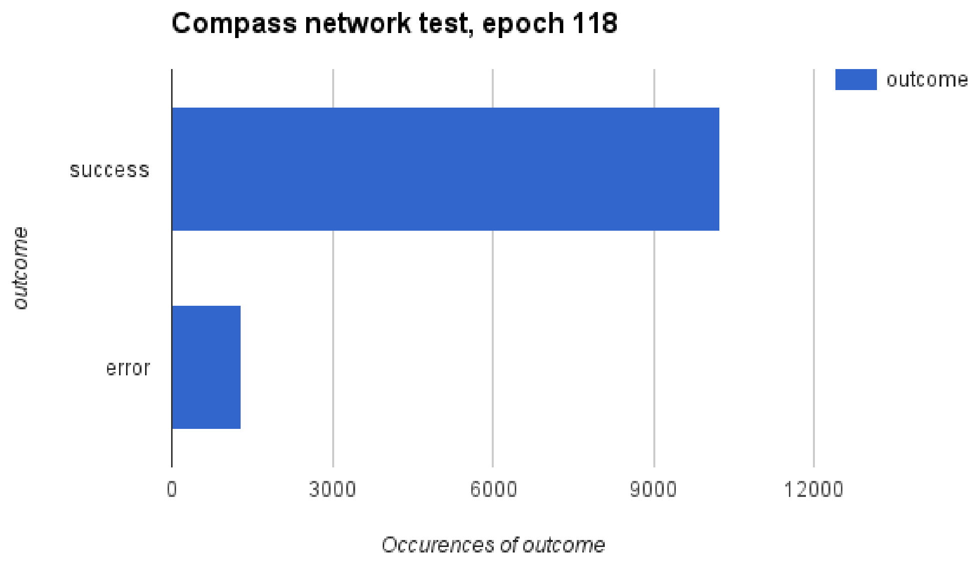


Figure 5.22: Test 3 magnetometer network test

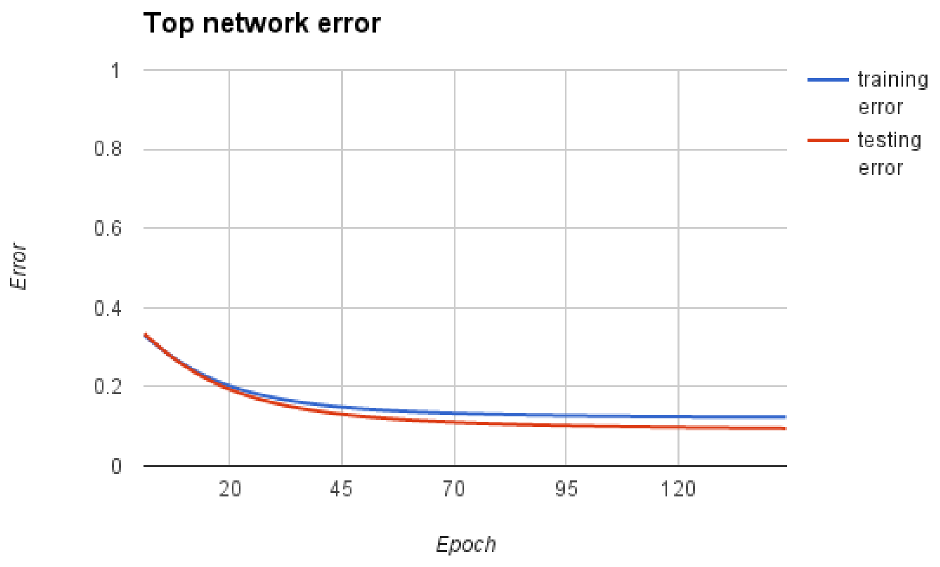


Figure 5.23: Test 3 top error

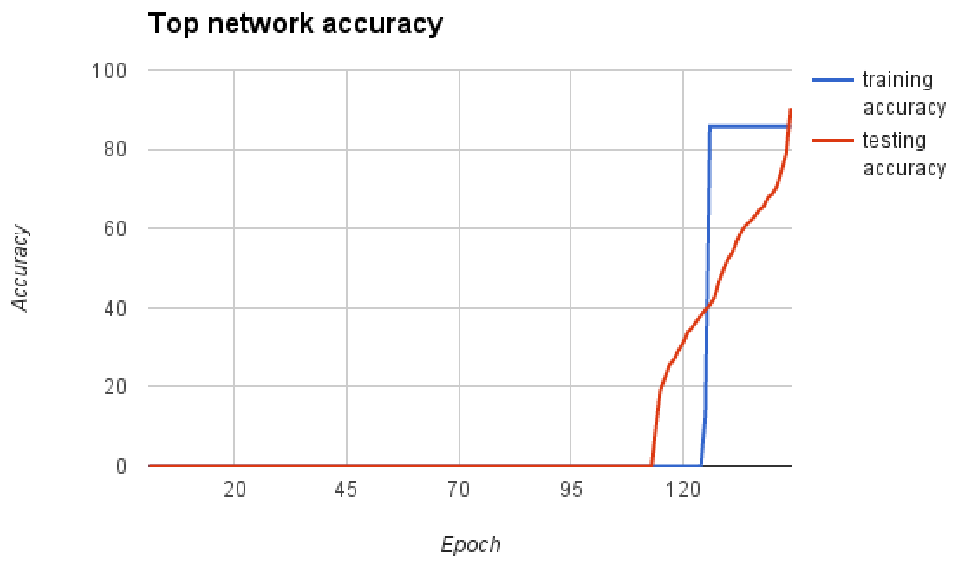


Figure 5.24: Test 3 top accuracy

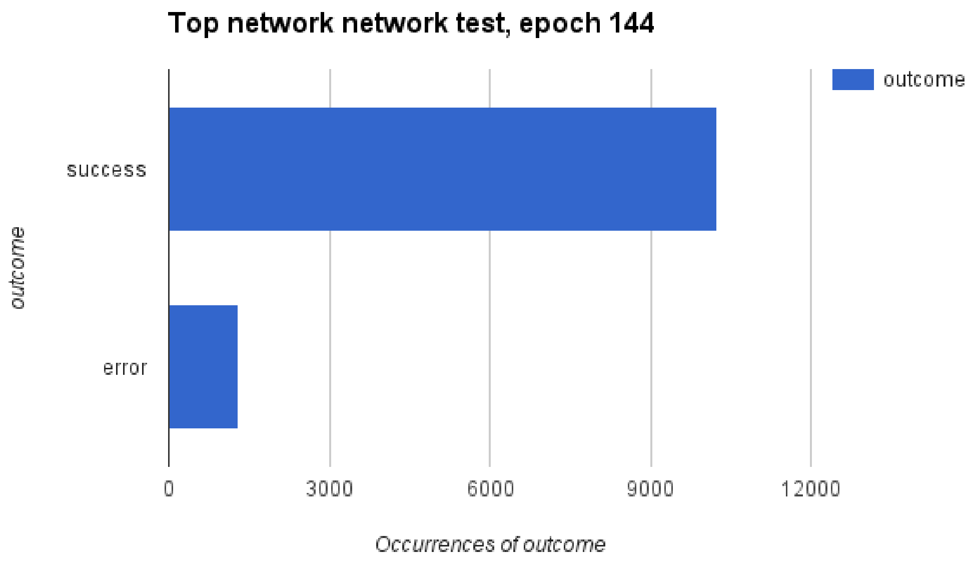


Figure 5.25: Test 3 top network test

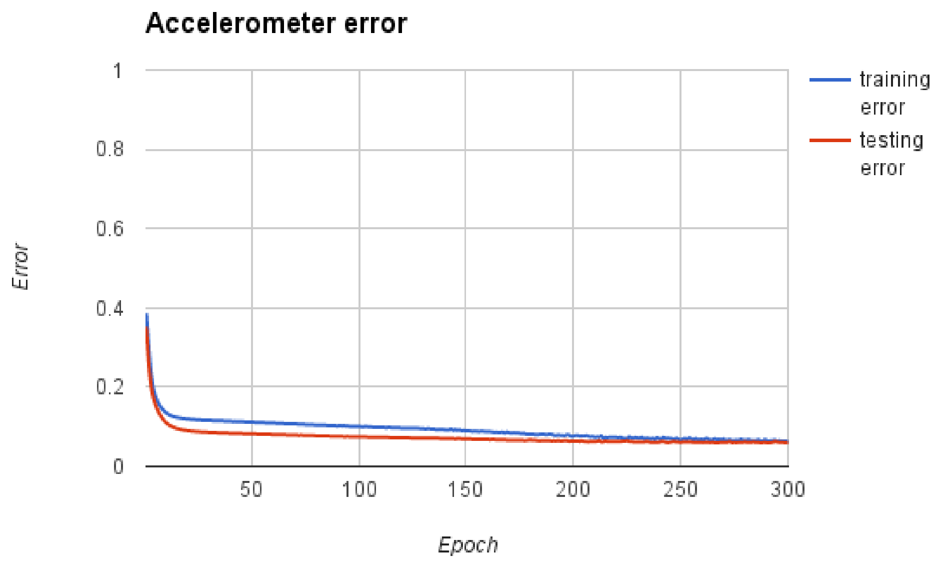


Figure 5.26: Test 4 accelerometer error

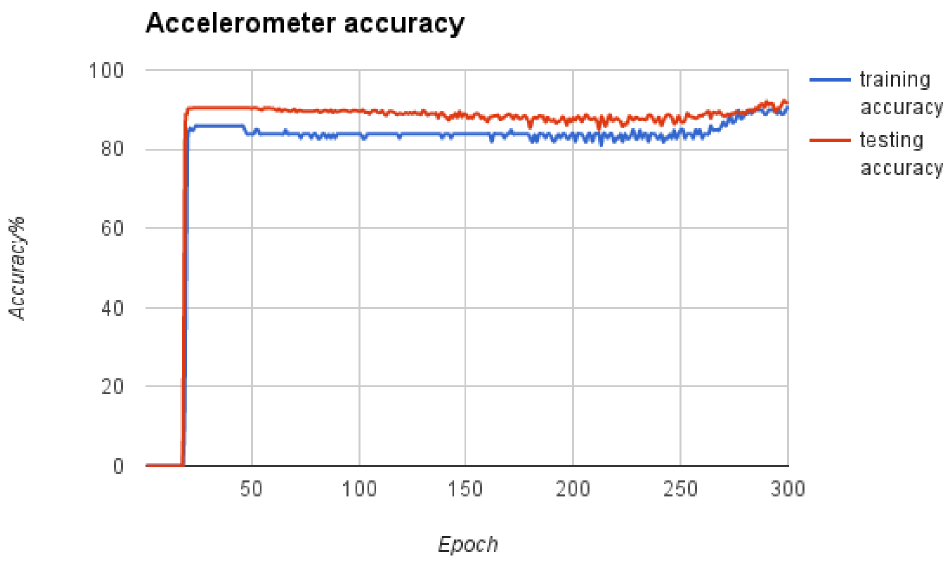


Figure 5.27: Test 4 accelerometer accuracy

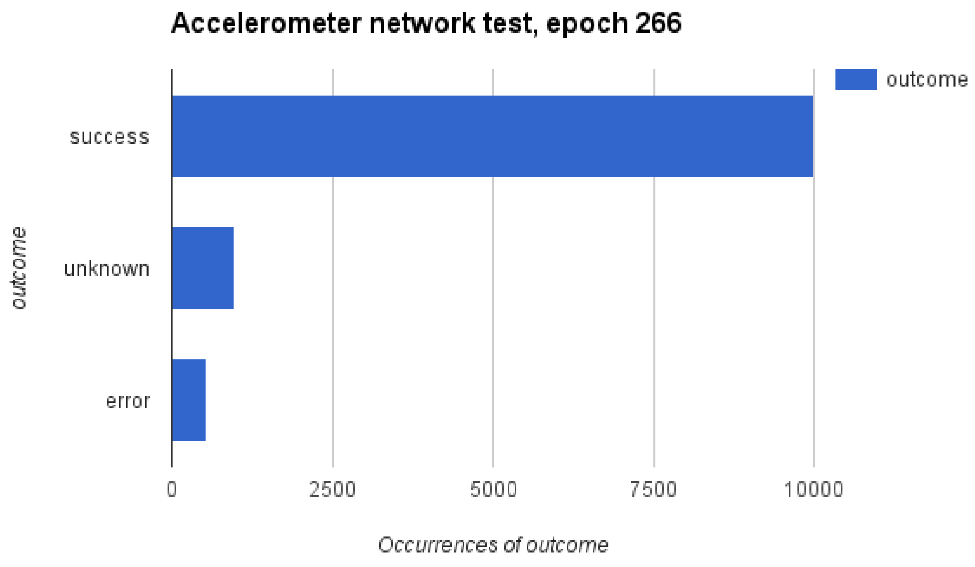


Figure 5.28: Test 4 accelerometer network test

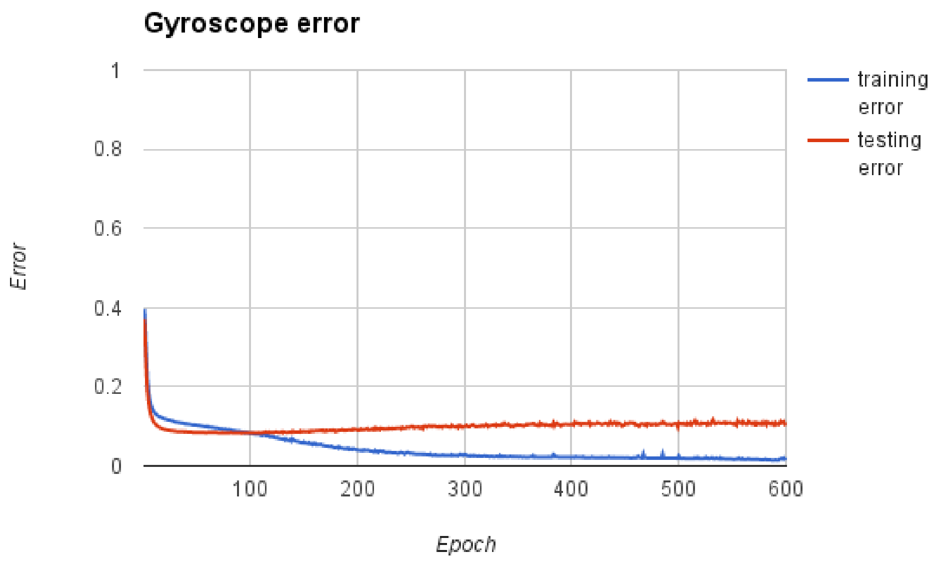


Figure 5.29: Test 4 gyroscope error

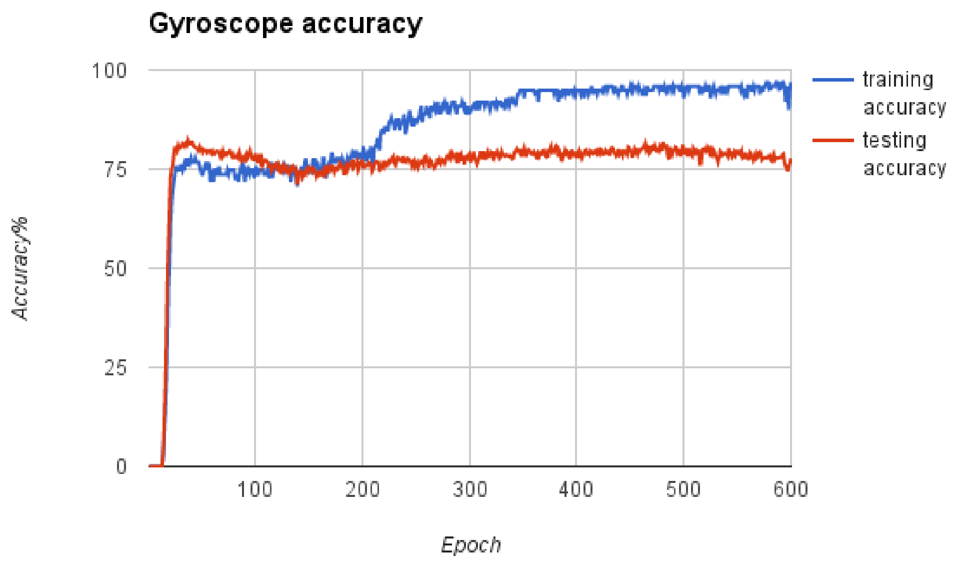


Figure 5.30: Test 4 gyroscope accuracy

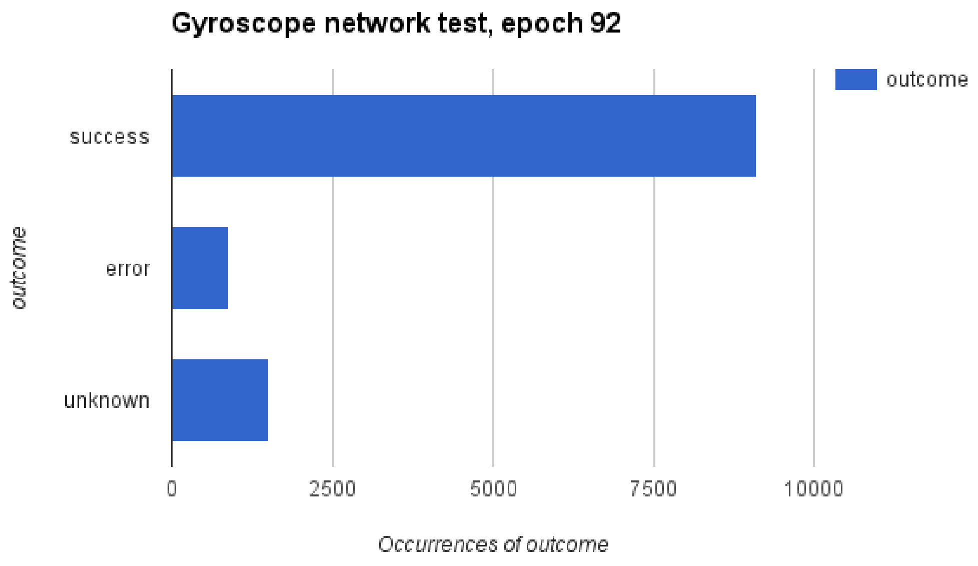


Figure 5.31: Test 4 gyroscope network test

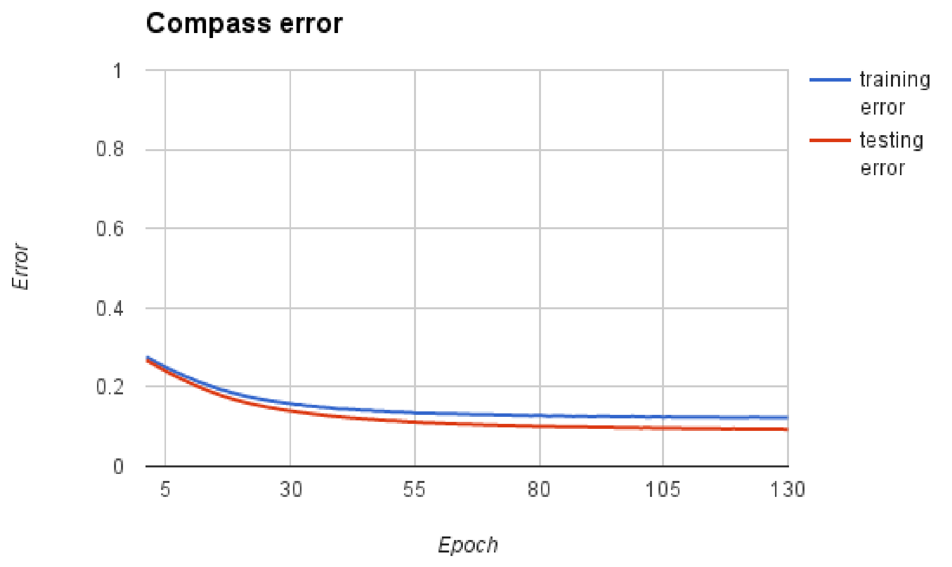


Figure 5.32: Test 4 magnetometer error

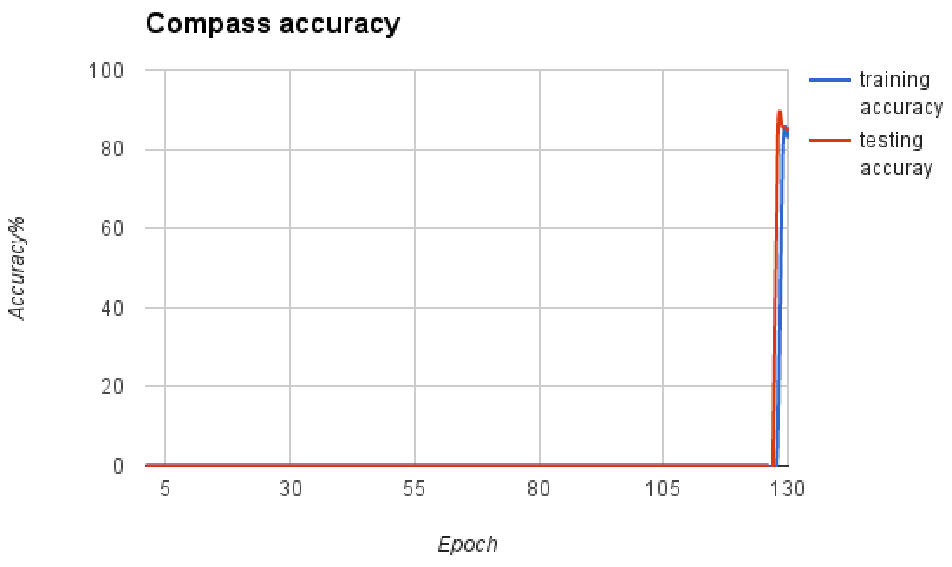


Figure 5.33: Test 4 magnetometer accuracy

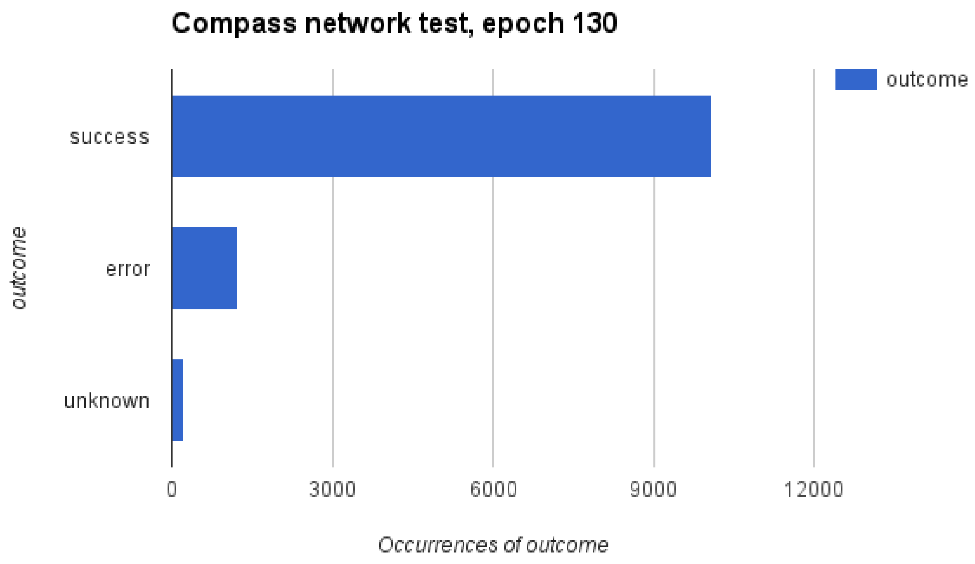


Figure 5.34: Test 4 magnetometer network test

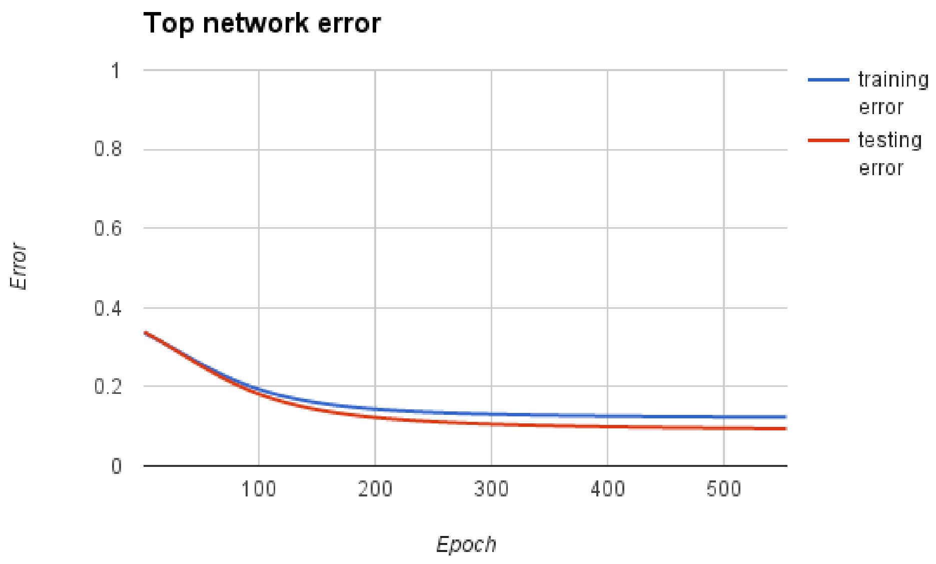


Figure 5.35: Test 4 top error

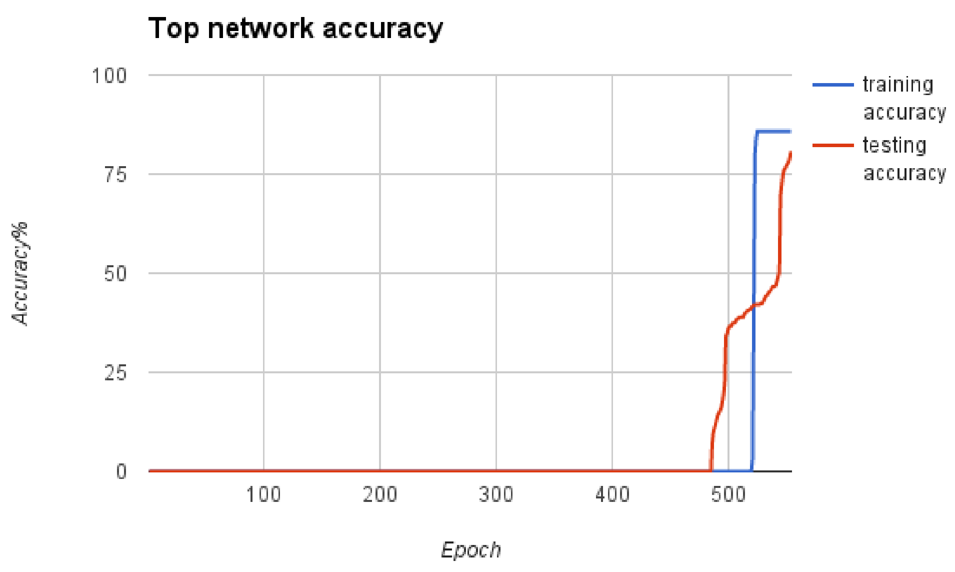


Figure 5.36: Test 4 top accuracy

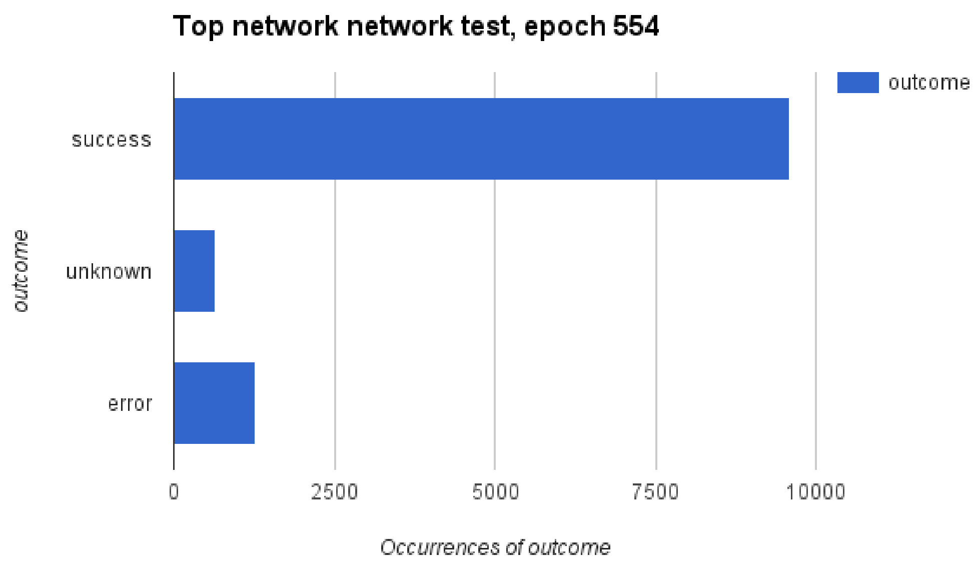


Figure 5.37: Test 4 top network test

5.1.4 Summary

Table 5.4 shows a summary of the network tests performed with the "All experiments" data set with at the epoch with the lowest error for the testing set. It is important to note that the table counts every output, not just the result. Therefore the *Accuracy percent* column is not indicative to the other columns, but rather represents the accuracy of the end results, not the individual output neurons.

Table 5.4: Summary of the network tests at best epoch

Test number	Network	Epoch	Successes	Errors	Unknowns	Accuracy percent	Test error
2	RefinedAccelerometer	244	10212	577	737	86,06	0,057638
	RefinedGyroscope	115	8520	768	2238	71,49	0,095971
	RefinedCompass	136	10186	1216	124	87,92	0,098995
3	RefinedAccelerometer	113	9689	1090	747	79,28	0,102948
	RefinedGyroscope	221	9618	1067	841	78,45	0,096587
	RefinedCompass	118	10240	1286	0	88,84	0,105003
	RefinedTopNetwork	144	10240	1286	0	88,84	0,104971
4	RefinedAccelerometer	266	9995	553	978	86,47	0,056876
	RefinedGyroscope	92	9115	893	1518	76,07	0,095555
	RefinedCompass	130	10080	1224	222	86,97	0,104124
	RefinedTopNetwork	554	9608	1268	650	77,88	0,104988

5.2 Discussion

5.2.1 The first test run

The very first test with a complete network, with the network being constructed of subnetworks, the subnetworks had identical settings: 16 inputs, two hidden layeres with 16 and 6 nodes within the respective layers and 2 output nodes which serves as the input nodes for the overarching network. The overarching network had the same number of hidden layers and the same number of nodes within them, aswell as two output nodes representing fall and not fall. Additionally there were no stop functionality in case of overfitting, hence I expected overfitting to occur. This was test was performed to see if the system developed would work, not for getting results, but it yielded interesting insight in the workings of the data. Figures 5.1 to 5.4 shows the accuracy and error done with these networks.

When it comes to the selected data sets for this first run I had only selected two exercises: number 1 and number 3, which is represented by the ready made collection "First testing run".

As can be seen in Figure 5.4 the training accuracy is vastly higher than the testing accuracy. As mentioned earlier overfitting was expected to be an issue for this first test, however this is likely due to the extreme complexity compared to the input of that part of the network. Essencially it has 6 input nodes, where the inputs range between 0 and 1. The large number of hidden neurons are therefore excessive in terms of analysing the inputs.

We can also see that the compass network does not really contribute at all with it's low accuracy and high errors with this configuration. I do however suspect that this trend will continue even when I start to refine the network settings.

One thing that must be mentioned, which I did not notice until very late in the refining of the network, is that the biases in the network were not being updated when I performed the first test run. Because of this the biases were always the same, and could hamper the learning. After running the same conditions again I would get roughly the same results.

5.2.2 Refining the network

The accelerometer network is one we often see capable of having an accuracy in the 80-90% range. Because of this I haven't changed the structure of this network. From *Test 2* to *Test 4* we can see it consistently hitting the target accuracy, see Figures 5.6, 5.15 and 5.27. A common trend we can see by looking at the figures of this network is that once they start increasing the initial accuracy it quickly climbs to high recognition rates. This is often reflected in the errors, see Figures 5.5, 5.14,and 5.26. Both *Test 2* and *Test 4* show this behaviour where the error quickly decreases and then slowly converges. *Test 3* is an annomaly amongst the two, however this is explained by the very low learning rate present when performing this test. Although the network tests performed in *Test 2* and *Test 4* are further from their target accuracies than *Test 3*, these two tests also have a lower test error. This can be attributed the longer training cycle which these two networks enjoy. I quickly performed a test on *Test 3* by changing the target accuracy from 80% to 90%, and I found that with this learning rate, the higher accuracy requirement could be obtained, however the difference in training time would not be worth the difference and risk overtraining the network, which sometimes would occur.

The gyroscope network was the most puzzling of the sub-networks to study. It never hit the target accuracies set in the tests, and had to be stopped early in every test. I attempted to use several different structures for the network: 1 hidden layer with different number of nodes; 2 layers with different number of nodes; and 3 layers with different number of nodes. Little did much to improve the errors and accuracies, nor my problem of overfitting. Therefore I kept the initial structure and went to tweak on the learning parameters. We do see improvements in the error between *Test 3* and *Test 4*. We do see from both Figure 5.8 and 5.29 that the network overlearns extremely quickly at higher learning rates, so with the current structure it is more reasonable to use a lower learning rate in order to get a better stopping point for the network.

In 5.2.1 I mentioned that I suspected that the compass would not contribute much to the network. After refining the learning settings and network structure I'm pleased to say I was wrong on my suspicious. After refinement we can see that the compass has consistently contributed with a high accuracy, see Table 5.4. It is however important to note that this network can quickly become overtrained, and needs to be stopped early manually. I have seen it manage to somehow overfit the testing set, at least it seemed that way when the testing set accuracy would be in the high 80%'s, but when running the "All experiments" data set the accuracy would be around 25%. Because of this it is recommended to run a low learning rate on this network and stop it early. It is also noteworthy that that this network typically has the highest number of errors to unknowns of the sub networks. So while the network has a high recognition, it also has many wrong predictions, as opposed to simply not knowing what action is being performed.

Lastly the top network is only present in *Test 3* and *Test 4*. In *Test 3* we can see the alarming similarity between the RefinedTopNetwork and RefinedCompass for this test. They have an identical number of successes, errors, unknowns and accuracy, with the sole differences being the epoch this was achieved and the test error. Looking into the output data for the tests for the two networks I can see that they are mostly identical. They have the same errors for the same data sets. This tells me that the top network can learn to lean too much on one of it's subnetwork's outputs, which is what I suspect happened here. Because of this I lowered the learning rate for *Test 4* to keep an eye on the error, but looking at both Figures 5.23 and 5.35 gives little indication on what's going on.

But when we look at the top network's accuracies we something interesting, see Figures 5.24 and 5.36: the testing accuracies starts climbing rapidly, before the testing accuracy. We also see that the training accuracy rockets up and then flatlines onwards. We can also see that the testing accuracy slows down a little around the time the training accuracy catches up, but it quickly gains steam afterwards. I haven't been able to find any litterature explaining this phenomena, however I suspect that it's learning to lean on one network's outputs, thus over-

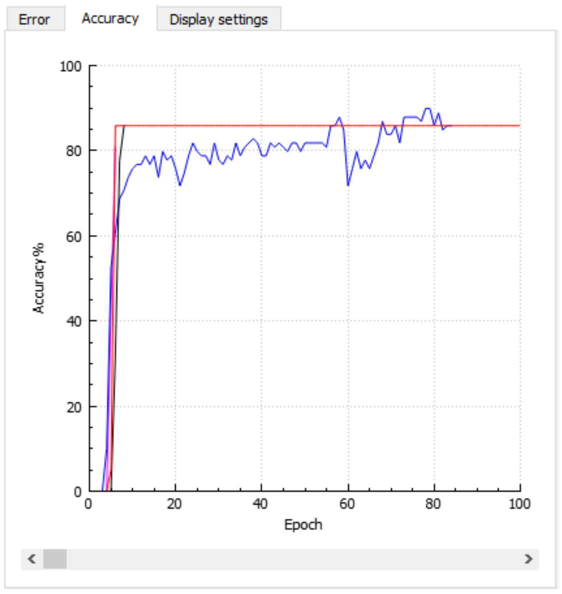


Figure 5.38: Top network flatline accuracy. Red - TopNetwork, blue - Gyroscope, black - Compass, magenta - accelerometer.

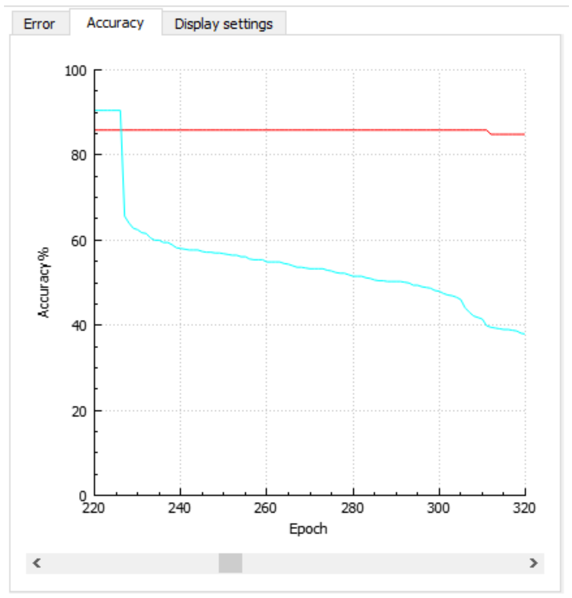


Figure 5.39: Top network overtraining on one network, accuracy. Red - training, cyan - testing

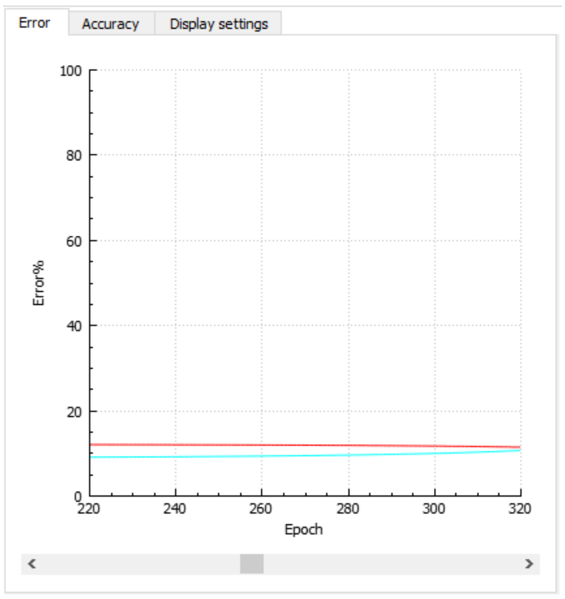


Figure 5.40: Top network overtraining on one network, error. Red - training, cyan - testing

fitting on that network. This would explain the sudden jump in training accuracy and why it flatlines the training accuracy, however because of the lower target accuracy of 80% the network stops training before the overfitting becomes too severe. The effects of this can be seen if we run the default learning parameters for the subnetworks, but raise the top network's target accuracy to 90%. We can then see that the network's training accuracy moves towards one of the sub-network's training accuracies and flatlines the training accuracy on it, see Figure 5.38. From there we can see the training error very slowly decreasing, while the testing error slowly increases until it reaches a breaking point, see Figures 5.39 and 5.40 and we can definitely see overfitting of the network. I afterwards stopped and ran the "All experiments" data set through it where it got the best results at epoch 28. By looking at Table 5.5 we can quickly discern which network the top network is leaning on. The testing and training accuracies between the top and compass networks are alarmingly similar, and the errors between the two aren't far away

Table 5.5: Top network overtraining test

Network	Epoch	Training Error	Training Accuracy	Testing Error	Testing Accuracy
RefinedTopNetwork	28	0,121549	85,858586	0,088184	90,476190
RefinedAccelerometer	6	0,121296	80,808081	0,090468	88,888889
RefinedGyroscope	20	0,079371	75,757576	0,079970	80,423280
RefinedCompass	8	0,121957	85,858586	0,090332	90,476190

from each other either. Running the “All experiments” data on both networks yield near identical results.

This means the top network requires more countermeasures against overfitting; or maybe it simply has the wrong structure; or the learning parameters aren’t optimal, as we do not see this exact scenario in *Test 4*, however the results in *Test 4* can be the result of variance within the neural network. This has not been further researched during this thesis. Instead, due to time constraints, I have used this potentially overtrained network, as it yields adequate testing accuracies with the entirety of the data colllection, however doing more experiments and implementing more data sets into the testing may reduce this accuracy.

Chapter 6

Conclusions and future work

6.1 Reviewing the objectives

The objective of this thesis was to demonstrate movement capturing and recognition by means of bluetooth sensors. This implied creating an experimental rig and software which can capture and store movement data. Futhermore a machine learning agent was to be developed in order to analyse and learn these activities.

A simple experimental rig has been created with one single CC2650 sensor and can be strapped around a person's waist for use. A modification of the accompanying software, originally by Texas Instruments, has been created which can capture, store and save these motions for analysis. This rig has then been used to test and record six different activities, repeated three times, by ten different individuals.

An artificial neural network has been developed as the machine learning agent and we have seen a higher recognition on falls against non-falls than the average commercial fall detection sensor of 75% [22], but the small sample size and limited variations in the activities is highly likely to skew the recognition rate. The accuracy of the top network can also be skewed by overlearning on a sub-network, which was discussed in detail in 5.2.2.

Through the results we have proven that it is possible to capture motions from bluetooth sensors and use this data to train an machine learning algorithm to recognise activities, or at the very least distinguish one activity from every other activity, with a high level of accuracy.

6.2 Observations of note made during the thesis

6.2.1 The inadequacy of the barometric pressure sensor

It was planned to use the barometric pressure sensor in conjunction with the other sensors in order to give a sense of broader vertical movement. This could for example help discern if a person is: moving in an elevator; moving up or down stairs; and many other activities. The barometric pressure sensor did however not deliver data of sufficient relevancy. It was deemed too inaccurate and it's implementation was therefore cut from the learning agent. The capture device does still capture and save the barometric data, however.

6.2.2 The gyroscope and magnetometer contributions to the accuracy

In 5.2.1 I made the claim that I suspected the magnetometer to be the least contributing sensor to the network's accuracy. However I found that the gyroscope is the least contributing sensor to the recognition, at least with the final structure used in this thesis. This was a surprise, as I expected the gyroscope to contribute a great deal together with the accelerometer to yield recognition. I speculate that the motions recorded by the gyroscope may be too similar to each other to gain a higher accuracy with the current structure. Another speculation is that the data resolution for the gyroscope is too low at a sampling frequency of 10 Hz.

I did not expect the magnetometer to gain this high recognition rate either. It could be caused by the preset locations the experiments were performed in, with the only direction variance being exercise 1 where the participant had free reign of movement within a gymmnastic hall. For every other exercise there was practically a preset direction for every activity. This warrants further study to conclude, but it is highly likely.

6.2.3 Top network can overtrain on one subnetwork

As discussed at length in 5.2.2 we can see that a top layer network can potentially overtrain on the sub-networks. It does this by overly emphasising the inputs from this network over the other sub-networks, and thus mimicing the results of said sub-network. While the results yielded for this report were deemed sufficient as proof of concept, I believe the networks created during this study to be fit for purpose and can be a good starting point, and therefore further study and development is necessary in order to use them for larger analysises.

6.3 Future work

Because of the nature of this project there are several ways to extend it. Not all plans and discoveries could be researched and implemented within the time constraints. Therefore I suggest the following tasks to research, in no particular order:

- Implementing more ways to combat overfitting (e.g: regularization; weight decay; other error functions; pruning; and/or more).
- Create a graphical representation of the network and it's connecting weights - makes it easier to analyse the network itself.

- Real time analysis - create a server-client relationship between the learning agent and the capturing device to analyse activities live (e.g live testing).
- Expand the number of activities.
- Genetic algorithm [28, 33] to optimise structure and learning parameters of the networks.
- Save the trained networks to files.
- Create a better pre-processing parser for data collections, aswell as change the network to use any data (the software can currently only read and analyse data from the modified BLE Sensor Tag application).
- Research the barometric pressure sensor and if it can be calibrated better, and if so implement it into the system.
- Implementing sensor configuration for the capture device (e.g disabling capture of certain sensors).

Bibliography

[1] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[2] K. Safi; F. Attal; S. Mohammed; M. Khalil; Y. Amirat. Physical activity recognition using inertial wearable sensors x2014; a review of supervised classification algorithms. In *2015 International Conference on Advances in Biomedical Engineering (ICABME)*, pages 313–316, Sept 2015.

[3] Ferhat Attal; Samer Mohammed; Mariam Debabrishvili. Physical human activity recognition using wearable sensors. *Sensors*, 1(15):31315–31338, 2015.

[4] Android Developers. Bluetooth low energy. `https://developer.android.com/guide/topics/connectivity/bluetooth-le.html`. Last accessed 20 June 2016.

[5] Christine Gulbrandsen. Høy dødsrisiko for de som faller. `http://forskning.no/aldring-forebyggende-helse-fysioterapi-helsepolitikk/2009/10/hoy-dodsrisiko-de-som-faller`. Last accessed 05 September 2016.

[6] Douglas M. Hawkins. The problem of overfitting. *Journal of Chemical Information and Modeling*, 44:1–12, 2004.

[7] JA Grisso; DF Schwarz; AR Wishner; B Weene; JH Holmes; and RL Sutton. Injuries in an elderly inner-city population. *Journal of the American Geriatics Society*, 38(12):1326–1331, 1990.

[8] Multiple. Owner Texas Instruments. Sensortag-20-android gitorious repository. https://git.ti.com/sensortag-20-android. Last accessed 14 July 2016.

[9] Texas Instruments. Cc2650 sensortag user's guide. http://processors.wiki.ti.com/index.php/CC2650_SensorTag_User's_Guide. Last accessed 05 September 2016.

[10] Inc. InvenSense. Mpu-9250 nine-axis (gyro + accelerometer + compass) mems motiontracking^{TM} device. https://www.invensense.com/products/motion-tracking/9-axis/mpu-9250/. Last accessed 05 September 2016.

[11] Ron Kohavi. Glossary of terms special issue on applications of machine learning and the knowledge discovery process. http://ai.stanford.edu/~ronnyk/glossary.html. Last accessed 27 December 2016.

[12] Link Labs. Bluetooth vs. bluetooth low energy: What's the difference? http://www.link-labs.com/bluetooth-vs-bluetooth-low-energy. Last accessed 19 June 2016.

[13] David Leverington. A basic introduction to feedforward backpropagation neural networksk. http://www.webpages.ttu.edu/dleverin/neural_network/neural_networks.html. Last accessed 27 December 2016.

[14] Lorin P. Maletsky; Junyi Sun; Nicholas A. Morton. Accuracy of an optical active-marker system to track the relative motion of rigid bodies. *Journal of Biomechanics*, 40(3):682–685, 2006.

[15] Multiple. Perceptron. https://en.wikipedia.org/wiki/Perceptron. Last accessed 25 December 2016.

[16] Multiple. Scale-invariant feature transform. https://en.wikipedia.org/wiki/Scale-invariant_feature_transform. Last accessed 27 December 2016.

[17] BBC News. Bluetooth rival unveiled by nokia. `http://news.bbc.co.uk/2/hi/technology/5403564.stm`. Last accessed 19 June 2016.

[18] Michael Nielsen. Neural networks and deep learning, a free online book. `http://neuralnetworksanddeeplearning.com/index.html`. Last accessed 27 December 2016.

[19] Stuart Russel; Peter Norvig. *Artificial Intelligence A Modern Approach*. Prentice Hall, 2010.

[20] Elvis Pfützenreuter. Bluetooth: Att and gatt. `https://epxx.co/artigos/bluetooth_gatt.html`. Last accessed 06 September 2016.

[21] Warren S. McCullochWalter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[22] Jeph Preece. The best fall detection sensors of 2016. `http://www.toptenreviews.com/health/senior-care/best-fall-detection-sensors/`. Last accessed 15 December 2016.

[23] Lorenzo L. Pesce; Richard M. Zur; Yulei Jiang and Karen Drukker. Noise injection for training artificial neural networks: A comparison with weight decay and early stopping. *Medical Physics*, 36(10):1929–1958, 2009.

[24] Stephen N Robinovitch; and Fabio Feldman. Video capture of the circumstances of falls in elderly people residing in long-term care: an observational study. *The Lancet*, 381(9860):47–54, 2012.

[25] Nitish Srivastava; Geoffrey Hinton; Alekx Krizhevsky; Ilya Sutskever; Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Reserach*, 15(1):1929–1958, 2014.

[26] Hector Allende; Claudio Moraga; Rodrigo Salas. Artificial neural networks in time series forecasting: A comparative analysis. *Kybernetika*, 38(6):685–707, 2002.

[27] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):535–554, 1959.

[28] Lothar M. Schmitt. Theory of genetic algorithms. *Theoretical Computer Science*, 259(1-2):1–61, 2001.

[29] Bosch Sensortec. Bmp280 barometric pressure sensors. `https://www.bosch-sensortec.com/bst/products/all_products/bmp280`. Last accessed 05 September 2016.

[30] Bosch Sensortec. Data sheet bmp280 digital pressure sensor. System document on Bosch's webpage, `https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMP280-DS001-12.pdf`, 2015.

[31] Leonid Sigal. Lecture 3 (marker-based) motion capture. Web lecture notes, `http://www.cs.cmu.edu/~yaser/Lecture-3-MarkerBasedMocap.pdf`, 2012.

[32] Gavin C. Cawley; Nicola L. C. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Reserach*, 11(1):2079–2107, 2010.

[33] Michael D. Vose. *The Simple Genetic Algorithm: Foundations and Theory.* MIT Press, Cambridge, MA, USA, 1998.

[34] Xingou Yu. Approaches and principles of fall detection for elderly and patient. *IEEE Intl. Conf. on e-Health Networking, Applications and Service*, 10(1):42–47, 2008.

Appendices

Appendix A

Thesis description document

UiT

THE ARCTIC UNIVERSITY OF NORWAY

Master of Science

Master thesis by

Stud. techn. Elisabeth Gangenes

Motion Capturing Machine Learning

UiT – Norges arktiske universitet

Faculty of Technology and Engineering Sciences

Computer Science - autumn 2016

The goal of this project is demonstrate movement capturing and recognition by means of bluetooth sensors. This implies that an experimental rig will be created that connect one or more bluetooth devices which can capture movement patterns of a person and develop an agent which can learn to recognize these movements. A smart phone will be used to record the data, while a desktop program will perform the machine learning. Examine the learning process

A literature study of relevant topics should be included.

Consequently the student should:

1. Create an application which can capture and store motion data for analysis.
2. Create a case of motions to capture with a suitable device or set of devices.
3. Establish a machine learning algorithm and pre-processing method to interpret the data of each sensor in conjunction with what it’s company sensors detect.
4. Run test cases on the system.
5. Document method, experimental rig and results from the experiment

General information

This master thesis should include:

- ❄ Preliminary work/literature study related to actual topic
  - A state-of-the-art investigation
  - An analysis of requirement specifications, definitions, design requirements, given standards or norms, guidelines and practical experience etc.
  - Description concerning limitations and size of the task/project
  - Estimated time schedule for the project/ thesis
- ❄ Selection & investigation of actual materials
- ❄ Development (creating a model or model concept)
- ❄ Experimental work (planned in the preliminary work/literature study part)
- ❄ Suggestion for future work/development

Preliminary work/literature study

After the task description has been distributed to the candidate a preliminary study should be completed within 4 weeks. It should include bullet points 1 and 2 in “The work shall include”, and a plan of the progress. The preliminary study may be submitted as a separate report or “natural” incorporated in the main thesis report. A plan of progress and a deviation report (gap report) can be added as an appendix to the thesis.

In any case the preliminary study report/part must be accepted by the supervisor before the student can continue with the rest of the master thesis. In the evaluation of this thesis emphasis will be placed on the thorough documentation of the work performed.

Reporting requirements

The thesis should be submitted as a research report and must include the following parts; Abstract, Introduction, Material & Methods, Results & Discussion, Conclusions, Acknowledgements, Bibliography, References and Appendices. Choices should be well documented with evidence, references, or logical arguments.

The candidate should in this thesis strive to make the report survey-able, testable, accessible, well written, and documented.

Materials which are developed during the project (thesis) such as software/codes or physical equipment are considered to be a part of this paper (thesis). Documentation for correct use of such information should be added, as far as possible, to this paper (thesis).

The text for this task should be added as an appendix to the report (thesis).

General project requirements

If the tasks or the problems are performed in close cooperation with an external company, the candidate should following the guidelines or other directives given by the management of the company.

The candidate does not have the authority to enter or access external companies’ information system, production equipment or likewise. If such should be necessary for solving the task in a satisfactory way a detailed permission should be given by the management in the company before any action are made.

Any travel cost, printing and phone cost must be covered by the candidate themselves, if and only if, this is not covered by an agreement between the candidate and the management in the enterprises.

If the candidate enters some unexpected problems or challenges during the work with the tasks and these will cause changes to the work plan, it should be addressed to the supervisor at the UiT or the person which is responsible, without any delay in time.

Submission requirements

This thesis should result in a final report with an electronic copy (i.e. CD/DVD, memory stick) of the report included appendices and necessary software codes, simulations and calculations. The final report with its appendices will be the basis for the evaluation and grading of the thesis. The report with all materials should be delivered in one signed loose leaf copy, together with three bound. If there is an external company that needs a copy of the thesis, the candidate must arrange this. A standard front page, which can be found on the Narvik University College internet site, should be used. Otherwise, refer to the "General guidelines for thesis" and the subject description for master thesis.

The final report with its appendices should be submitted no later than the decided final date. The final report should be delivered to the adviser at the office of the IVT Faculty at the UiT.

Date of distributing the task:

Date for submission (deadline):

Supervisor UiT - IVT	Professor Bernt A. Bremdal Phone: (+47) 90061173 e-mail: bernt.a.bremdal@uit.no
Co-supervisor UiT - IVT	Asbjørn Danielsen
Candidate contact information	Elisabeth R. Gangenes e-mail: elisabeth.dimby@gmail.com

Appendix B

Abbreviations and acronyms

- API: Application Programming Interface
- ATT: Attribute Protocol
- BLE: Bluetooth Low Energy
- CC2650: CC2650 SimpleLink multi-standard 2.4 GHz ultra-low power wireless MCU
- CSV. Comma-separated Values
- DMP: Digital Motion Processor
- GATT: Generic Attribute Profile
- IDE: Integrated Development Environment
- IR: Infra-red
- LE: Low Energy
- LED: Light Emitting Diode

- k-NN: k-Nearest Neighbour
- Mbps: Megabit per second
- MSE: Mean Square Error
- SVM: Support Vector Machine
- UiT: Universitetet i Tromsø
- UUID: Universally Unique Identifier

Appendix C

List of figures

List of Figures

Appendix D

List of tables

List of Tables