An Automatic Detection of Rugby Union Player Motions Using Time Series Analysis

Nicole Catherine Donnelly

September 2017

Dissertation submitted in partial fulfilment for the degree of
Master of Science in Big Data

Computing Science and Mathematics
University of Stirling

Abstract

Advances in wearable technology have proven to be a huge benefit in the professional sports industry. Athlete behaviour and rehabilitation can now be monitored through the data produced by wearable sensors. The ability to classify a professional athlete's movement will reap benefits for the manager of any team when it comes to improving team strategy on match day, creating a more refined training routine or assisting in rehabilitation. Global Rugby Network aims to have these services available to professional and amateur teams across the globe to assist any coach.

Through the use of deep learning techniques, basic motions that an athlete would perform were classified. This was done by using a Long Short-Term Memory recurrent neural network. Signal Processing techniques were employed beforehand to improve the performance of the classifier.

A successful result was achieved with final and best accuracy measuring reaching 80.4%.

Attestation

I understand the nature of plagiarism, and I am aware of the University's policy on this.

I certify that this dissertation reports original work by me during my University project except for the following:

- The data collection and data labelling activities discussed in section 3 was carried out by Adam Rennie, a University of Strathclyde student also completing a summer placement with Global Rugby Network.
- The time slicing of the data discussed in section 3 was carried out and the sliding windows function written by Kostas Chartomatzis, a data scientist at Global Rugby Network.
- The code discussed in section 6 was largely taken from [73] and used in accordance with the licence supplied.
- The code discussed in Section 5 was developed by me during a vacation placement with the collaborating company.

Signature **Date**

Acknowledgements

I would first like to express my thanks to my thesis supervisor Professor Carron Shankland of the University of Stirling. She consistently allowed this paper to be my own work, but steered me in the right the direction whenever she felt I needed it. I must also express my gratitude towards Prof. Shankland for helping me take steps to overcome my imposter phenomenon, and for inspiring me in my pursuit of a career with her work in the technology field as a female.

I would also like to acknowledge the time spent and contribution of Professor Amir Hussain as the second reader of this thesis.

I am also indebted to the Data Lab for paying my tuition fees and making it possible to undertake the M.Sc. Big Data. Thank you to MBN Solutions for working in collaboration with the Data Lab and assisting me in finding this placement opportunity.

Thanks are also due to Global Rugby Network for affording me with an opportunity for learning and professional development. I consider myself very lucky for this opportunity and being able to be a part of this very exciting research project. Special thanks to Stefan Raue (CTO) for his continuous support and guidance and Kostas Chartomatzis (Data Scientist) for the many tasks he assisted me with.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Explanation
ANN	Artificial Neural Network
BN	Bayesian Networks
CNN	Convolutional Neural Network
CRISP-DM	Cross Industry Standard Process for Data Mining
CSV	Comma Separated Value
DAG	Directed Acyclic Graph
DFT	Discrete Fourier Transform
DSP	Digital Signal Processing
EC2	Amazon Elastic Compute Cloud
FFNN	Feedforward Neural Network
FFT	Fast Fourier Transform
FN	False Negative
FP	False Positive
GPS	Global Positioning System
GRN	Global Rugby Network
HAR	Human Activity Recognition
HMM	Hidden Markov Models
IMU	Inertial Measurement Unit
LDA	Linear Discriminant Analysis
LSTM	Long Short-Term Memory
MEMS	Micro-Electro-Mechanical Systems
PCA	Principal Component Analysis
QDA	Quadratic Discriminant Analysis
RNN	Recurrent Neural Network
SVM	Support Vector Machine
TN	True Negative
TP	True Positive
(1) G	Unit of Acceleration, equal to the standard value of the Earth's gravity = 9.8 ms^{-2}
°/s	Degrees per second

1 Introduction

"Winning in team sports has always been a function of superior ownership, front offices and coaching." [1] Moneyball: The Art of Winning and Unfair Game, was a book released in 2003 documenting Billy Beane, the manager of the Oakland Athletics baseball team, and his use of sabermetrics. This number one best seller chronicles his discovery of the secret to success in what can be described as the imperfect science of baseball player evaluation. [2]

Since the release of this book (that was later made into a box office hit movie) the popularity of data-driven decision making in the sports industry has increased exponentially; spreading to all professional team sports. Player tracking technologies have been described as a "game changer" and the "future of sport"; they are set to completely change the field as technology does for any industry.[3][4] With the Sporting Industry being a £20bn a year industry in the UK alone, [5] player performance, and using technology to monitor it, has never been of more interest.

1.1 Background and Context

As the market for wearable technology continues to explode and venture into new niches, it naturally progressed into professional sports. These wearable devices produce a "tsunami of data", the implications of which are beginning to be used in data science applications. [6]

This mountain of data can be overwhelming to a manager or coach of a professional team. Many different applications can be constructed using data such as; activity recognition, concussion detection, real-time statistics on player's speed, acceleration and heart-rate. Data analytics can be consulted to influence coaches' decisions yielding better results on the field.

Motion Sensor Classification is an important and relevant technology in persuasive computing as it can be applied to many real-life, human centric problems such as healthcare and rehabilitation. The substantially large datasets produced can be manipulated to develop a classifier, which allows for data-driven decisions to be made. Nowadays, technology for performance enhancement is a necessity for any competitive sport.

The objective of this classifier was conceived by Global Rugby Network (GRN); a software-driven start-up company based in Glasgow. GRN's core product is free team and performance management software. Amateur teams and emerging rugby nations have great difficulty in accessing professional team management, coaching and performance software. GRN is an innovative solution to this problem, allowing teams, coaches and unions to grow in rugby and become more successful. [7]

The launch of GRN's wearable performance tracking device has enabled the company to set out a road-map for the development of performance analytics. The company's initial focus will be placed upon descriptive statistics, such as speed, performance zones, impact, and positioning of individual players; all obtained through the data measured by the device.

1.2 Scope and Objectives

An exploratory investigation was carried out in the application of Deep Learning techniques on human data to produce a classification system. This research activity is geared towards using the substantial and increasing volume of player performance data to train and deploy a machine learning model. The model will aim to achieve the capability of classifying basic human motions. The movements selected for classification will commonly appear in a rugby match or training session. This model (and results of later work) is aimed towards advanced analysis, such as in-detail player performance analysis, group strategy analysis, and injury prevention through player training/game load monitoring.

Within this framework, this project work was carried out as an exploratory research into predictive analytics. The scope of the work was defined as follows:

- explore raw sensor data from existing trials (London/Paris)
- apply signal processing techniques to detect outliers/noise within the data, and interpolate with suitable values
- contribute to the design of a controlled data capturing process
- implement a prototypical data processing pipeline
- research/identify a suitable machine learning technique
- implement proposed machine learning model
- obtain optimal configuration of implemented model based on predictive power

The intention of this research is to use the results as a foundation for future research into group motion/activity detection, injury prevention, and automatic video annotations. The work undertaken and final results are vital for improving the GRN software and hardware marketed to new and existing customers. After a sufficient amount of work with good results has been undertaken in this application area, the output will be a tailored application capable of classifying basic and more complex movements that could generate commercial interest and drive the overall sales and business development of GRN.

1.3 Achievements

Developing an algorithm that has the ability to detect context from noisy and ambiguous sensor data is the key difficulty in creating any robust computer application. Handling the subsequently large datasets produced from these sensors adds to the level of difficulty. Research of the advanced signal processing techniques and the implementation to improve the performance of the classifier has been an extremely challenging task. A steep learning curve had to be overcome in a short time frame, which included being able to visualise the signal and understand the meaning of its different aspects. Ensuring that accurate, high quality data would be implemented for training the classifier fell into this branch.

The data involved in this research and GRN's existing data is stored in csv files and not stored in a database as of yet. Being able to manage the data without a database was difficult at times and to overcome this challenge the Pandas library available for Python was used. It enabled reading of multiple csv files in an efficient manner. To achieve this, a basic knowledge of Python programming language had to be improved very quickly which included learning the data container and merging features of the Pandas library.

This research has met all objectives presented above, and the accompanying documentation provides a clear and concise thought process in order to understand and replicate this work. This will allow the company to make modifications to different steps in order to improve the performance. However, the results obtained have shown this solution to the motion sensor classification problem is a very promising foundation for future work.

This implementation can be extended to consider future data sources, different signal processing steps or alternatively a different algorithm for classifying movements.

1.4 Overview of Dissertation

This report is organised into chapters charting the development process of the Motion Sensor Classifier.

Section 2 – *State of the Art* provides an overview to the research carried out in order to fulfil the brief outlined. A brief passage about the significance and relevance of technology within sports will introduce this section. The background and current research available in motion sensor classification field will be summarised to provide sufficient understanding. The classification methods unearthed within this research will be discussed, with a particular focus upon the techniques used within the implementation of the classifier. Lastly, the Programming language and toolkit used will be presented.

Section 3 – *Design and Approach* presents an outline of the routine of activities used to collect the training data. A brief overview of all the stages throughout the project will be provided in the Methodology.

Section 4 – *Data Acquisition* presents detail on the player monitoring device used to capture the data which will be presented along with a description of the data it generates.

Section 5 – *Signal Processing* exhibits the pre-processing stage of the project. The various techniques used for this will be discussed in addition to others that have been researched. The reasoning behind the chosen techniques is discussed. The data cleaning (pre-processing), data segmentation and feature extraction steps are covered in this section.

Section 6 – *Neural Network Design* will discuss the various choices made for the LSTM classification algorithm in addition to the final step taken to prepare the data for the classifier.

Section 7 – *Analysis & Results* presents a clear summary of the different results achieved by varying different hyperparameters and features before discussing the analysis of the optimal result achieved by the model.

Section 8 – *Critical Evaluation* will discuss the limitations and challenges incurred in the project and will discuss future work and recommendations for data collection, a database and algorithm improvements that could be investigated.

Section 9 – *Conclusion* will summarise what was achieved in the project and re-iterate the configurations of the best result achieved.

2 State-of-The-Art

"The world's most valuable resource is no longer oil, but data." [8] The role of data in many industries has increased exponentially in the recent years, including professional sports. Machine learning contributes to data-driven decision making in sports and allows numerous patterns to be detected which the human eye alone cannot achieve. [9]

2.1 The significance of technology in sport

Human activity recognition and classifying motions is an emerging field of research, stemming from larger fields of ubiquitous computing. It is becoming a popular area of interest in deep learning through the use of inertial movement sensors. It has many applications which range from healthcare and rehabilitation to smart assistive technology; [10] all of which can be easily integrated to use in professional sports. The benefits of tracking how far and how fast an individual runs pay off quickly for a coach allowing them to focus more on developing the team. From analysed live data, a coach can instantly see if players are adhering to restrictions given on effort level in training; for example, training at only 50% effort level the day before a game to preserve strength. [11] In combination with other metrics such as hormone levels, a coach can monitor the effect jet lag has on a player's ability to perform; which for a sport like rugby (that travels to multiple locations during a season) could be highly useful. [12] Player tracking can be used to monitor fatigue levels during a game allowing coaches to make informed data-driven decisions about which players to send off and draft in. The data can determine if continuing to play is within the player's range of capabilities and strength or the risk of injury is increased. [13] The applications of a well-constructed motion sensor classifier can be extended to not only sports, but also used in military training applications and in health and education.

The combination of raw talent of an individual player and scientific analysis from assistive technology make for a more competitive and compelling sport.

2.2 Machine Learning

Pattern recognition is the theory that computers have the ability to learn specific tasks without being explicitly programmed for them. Researches in the field of Artificial Intelligence challenged this to see if computers could learn from data. This theory was intriguing as these models should have the functionality of independently adapting to new data from the learned data whilst producing reliable results. This technique (that is growing in popularity) has become known as Machine Learning. [14]

Rich data sources are now readily available to build problem solving models which can be integrated into working software to support products in high demand across industries. [15] It is important to consider the data available when selecting a machine learning model to use; moreover, the first decision is if supervised or unsupervised learning techniques will be applied.

2.2.1 Supervised Learning

Supervised Learning is the most commonly used method in machine learning applications. It entails the data consisting of a number of input variables and a single output variable for that instance of data; i.e., labelled data. This means the correct outcome for the variable is known and the role of the algorithm is to learn and iteratively make correct predictions on an instance of data. Supervised learning problems can be further grouped into classification and regression. Classification is implemented when a variable has a nominal output and the aim of the algorithm is to correctly predict a class. Regression tasks are used on numerical variables to achieve the ability of making a prediction or forecast. [16]

2.2.2 Unsupervised learning

Unsupervised is the exact inverse, it involves only input data with no subsequent output variables. An algorithm needs to discover and present the underlying structure of the data. There is no correct answer for the model to achieve and the inherent structure of the data is discovered without a "teacher", hence the name unsupervised. There is fewer use cases of unsupervised learning due to the level of complexity required to implement. [17] The two main branches of unsupervised learning are clustering and association. Clustering involves discovering groupings in the data meanwhile association is a rule based technique that uncovers a rule describing large portions of the data. [16]

When there is a mixture of labelled and unlabelled data a balance between supervised and unsupervised algorithms can be applied; this method is conversely known as semi-supervised learning. The motivation for this is that high quality labelled data can often be costly to generate meanwhile unlabelled data tends not to be. [18]

2.3 Literature Review of Related Work

Traditionally, wearable sensors have been used for motion capture and human activity recognition (HAR). This work has often involved a subject being confined to a laboratory or a similar environment with motion capture equipment, which is both restrictive and invasive. This type of research has widely investigated HAR based on vision data and 3D data. In the past decade or so, the interest has been placed upon Inertial Measurement Units (IMUs); a

wearable device that provides non-intrusive monitoring. Technical advances have reduced prices of sensors capable of capturing the input data necessary for this type of classification.

The OPPORTUNITY challenge was run in 2011 with a view to recognising activities in a home environment. It is a well-known application within the HAR technology field. The highest accuracy rate achieved was 88% upon the recognition of 17 gestures. [19] The conclusion was the performance of activity recognition will need to be improved upon before addressing more complex problems such as activity diarisation.

An extensive survey conducted by Avci et al. documents prior work on HAR using inertial sensors and an outline of pre-processing, data segmentation, feature extraction approaches, dimensionality reduction and classification techniques. Additionally, the survey illustrates the range of different applications such as healthcare, sports and wellbeing. It was observed that prior work in these areas typically involves gathering sensor information and post-processing as a separate offline step. Therefore, it concludes that performing activity recognition in real-time "remains an open research question". [20] A dataset of reduced or selected features is used as inputs for the classification techniques detailed in the survey, the results of which are shown below in Table 1. The classification methods displayed are expanded upon with related work in the motion sensor classification and human activity recognition fields of research.

Classification Model	Accuracy Rate (%)
Naïve Bayes	83.97
Decision Tables	46.75
Decision Trees	90.8
Nearest Neighbour	91
Support Vector Machine	87.36
Hidden Markov Models	90
Gaussian Mixture Models	88.76
Artificial Neural Networks	95

Table 1. Classification models and respective accuracy rate detailed in [20]

2.3.1 Motion Sensor Classification Techniques

Bayesian-based approaches for classifying include Bayesian Networks (BN) and Naïve Bayes both of which are simple, probabilistic classifiers based upon applying Bayes theorem. Naïve Bayes does so with strong, naïve independence assumptions between features whilst a Bayesian (belief) Network is a graphical based model that represents a set of random variables and their conditional dependencies via a Directed Acyclic Graph (DAG). There has been previous research using Bayesian methods in motion capture/HAR field of interest such as the work detailed in [21]. This involved mounting a MEMS IMU onto the belt of a participant to

gather data illustrating 7 activities. Bayesian Techniques are compared for this real-time classification; Bayesian Networks (BN), Inference in Static BN, Dynamic BN and Inference in Dynamic BN with a grid based filter. In [22], decision trees, Naïve Bayes and Naïve Bayes with Principal Component Analysis (PCA) in combination with 19 features extracted from time, frequency and spatial domains were used to classify five activities (walking, running, cycling, driving and sports). This work achieved a 72.3% accuracy measure.

A Decision Tree is a decision support tool that uses a tree like structure to model different outcomes. Random Forrest is an ensemble learning method for classification, which extends from decision trees. It operates by constructing a multitude of decision trees and the output class is the mode of the classes. Research conducted by S. A. Rawashdeh et al. proposed using motion capture for preventing shoulder injuries in overhead sports. A decision tree approach was adapted for building a classifier, with 86% accuracy being achieved. [23] In [24], a low-pass filter was used to remove outliers and the Euclidean magnitude used to combine signals into a single magnitude as pre-processing and feature work. A Random Forrest classifier is implemented to reach a performance of 83.49%.

The very extensive study conducted by C. Y. Yong et al. [25] also involves a sensor attached to the upper forearm to classify walking, jogging and throwing. PCA-K-means, Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), Naïve Bayes Gaussian Kernel, Naïve Bayes Kernel Density and Decision Tree are the classification algorithms investigated. The accuracy measures achieved were 94.67%, 64.33%, 77.0%, 75.0%, 86.0%, 94.67% respectively. The PCA-K-Means classifier was presented as the most successful by classifying all three motions with the shortest period, highest accuracy and lowest errors; appropriate for processing large sets of data.

RecoFit is a wearable sensor that measures tri-axial accelerometer and gyroscope data from the upper forearm. The data is smoothed using a low-pass Butterworth filter and sliding windows to extract statistical features. A multi-class Support Vector Machine (SVM) is used to achieve the classifications. [26]

Hidden Markov Models (HMM) are a statistical model based on the Markov chain process; the system being modelled is assumed to be a Markov process with unobserved (hidden) states. [27] describes the use of HMM as multi-class classifier. A moving average filter and sliding windows have been used as pre-processing steps to improve the accuracy rate, with the model achieving up to 93.39%. In [28] the classification algorithm is a hybrid mixture between Gaussian mixture and hidden Markov models. A sliding window and FFT have been applied to the data, succeeding in a good classification result. However, overfitting was a concern expressed by the authors of the paper.

There are various neural networks that can be implemented for classifying motions or activities. The work studied by L. Bao et al. [29] involved a multilayer feedforward neural network (FFNN) to classify 8 different activities. It achieved an impressive accuracy measure of 95% correctly classified. Similarly, the research carried out in [30] achieves a very successful 94% accuracy rate. It documents a continuous time recurrent neural network (which is a more general predictor), implemented on an isolated dataset to deal with the classification of 8 gestures. In [31], a forearm band, *PUSH* (similar to *RecoFit*), is used to obtain exercise motion data. An approach for classifying large scale wearable sensor data of exercise movements achieving an accuracy rate of 92.14% is demonstrated using a Convolutional Neural Network (CNN). In [32], five activities (running, walking, lying, standing and sitting) were classified using a multi-layer perceptron, achieving 91.1% accuracy. In this work 13 statistical features from time and frequency domains were utilised. Future work intends to investigate classification without segmentation by introducing deep CNN, or alternatively using LSTM (Long Short-Term Memory) by combining neural networks for time-series data. The benefits of deep learning approaches to classifying human motion based on CNN and LSTM architecture approaches are presented in [33]. The findings from this analysis support the hypothesis that an advantage of using LSTM-based models is the ability to learn the temporal feature dynamics, which CNNs are not fully capable of modelling. Two well-known publicly available datasets with the HAR field, *OPPORTUNITY* and *Skoda*, are used for implementation. The LSTM architecture approach was able to achieve an F1-score of 0.958 (see 2.5.4 below).

2.3.2 Classifier Chosen for Implementation

Human Motions are comprised of complex sequences of motor movements which will often have large variability between movements. This is a very challenging problem for any classifier to overcome. Conventionally, human activity related tasks have been solved by using heuristic processes to obtain engineered features which have always been a lengthy process. This approach was dominant in any field of recognition until deep learning presented improved performance results, better than the carefully crafted, complex and time-consuming feature detectors.

Achieving a solution to this problem is motivated by two requirements: attaining successful recognition accuracy and decreasing reliance on engineered features to address increasingly complex recognition problems. Feature engineering is a time consuming yet fundamental part in the workflow of a machine learning project. Deep learning techniques allow raw data to be used, as features will be automatically discovered and created by a neural network during training. However, this does not mean that data pre-processing and feature engineering methods are obsolete; there is simply not as much time-consuming work involved.

A recurrent neural network (RNN) is well-suited for handling sequence dependent (time-series) data. It is designed to recognise patterns in sequences of data, such as recognising a motion from a signal. Other applications of RNN include recognising text, handwriting or spoken word; all recognising a pattern in a given sequence. RNN in a network enables the higher level temporal features to be learned effectively. From the related work discussed above, immediately an LSTM network appeared to be favourable – partly due to the high accuracy measures detailed. The suitability of the RNN meant that the LSTM network would be used for the motion sensor classifier. It has the ability to learn from experience in order to process, predict and classify time-series. Sequences containing patterns of unknown length are suited to the LSTM due to the ability of the network's long term memory. It has been proven that very large architectures can be successfully trained using LSTM.

The reasoning above and the success shown in previous research demonstrates that an LSTM is extremely suitable for the motion sensor classifier at GRN. The architecture of this Neural Network is discussed below.

2.4 Neural Network Architecture

A neural network can be defined as a "beautifully biologically-inspired programming paradigm which enables a computer to learn from observational data." [34] As the name suggests, neural networks are inspired by how the biological nervous system handles information. It is modelled after the brain due to these two key aspects:

1. The network acquires knowledge through a learning process
2. Synaptic weights are interconnected strengths in network that store knowledge

Neural networks are powerful for pattern classification and are at the base of deep learning techniques. The fundamentals of recurrent neural networks are outlined by beginning with the more well-known feedforward neural network to provide a clearer explanation. In particular, those built on LSTM units.

2.4.1 From Feedforward to Recurrent Neural Networks

An artificial feedforward neural network was the first and simplest type of neural network. It is a computational model that processes information through a series of interconnected computational nodes (synaptic weights). The synaptic weights are grouped into layers and communicate directly with each other by using weighted connections. The model is loosely defined by nodes and connections. The information is fed into the network at the input layer and, using supervised learning, is transformed into an output (the predicted class). The hidden

layers are where the network is trained until it minimises the error it makes when predicting classes. Figure 1 demonstrates the structure of a basic FFNN with three hidden layers.

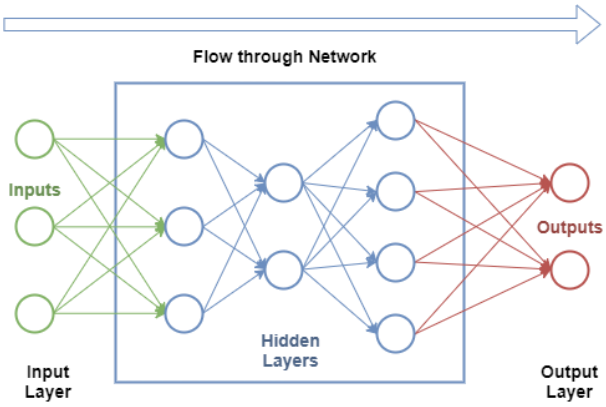


Figure 1. Architecture of a feedforward neural network

A FFNN that is extended to include feedback connections is known as a recurrent neural network. Both of these networks are named after the way they channel information, by performing a series of mathematical operations at the nodes of the network. One feeds information straight through (feedforward), whilst the other cycles it through a loop (recurrent). RNN is very adaptable for sequence data, which includes time-series predictions. [35]

The RNN has two sources of inputs; the current input instance and what it has recently observed from past examples (the outcome at time step *t-1* will affect the decision at time step *t*). In a FFNN the hidden layers do not interconnect with nodes in the same layer, only the nodes in the subsequent layer. Figure 2 below highlights the difference in structure between the FFNN and RNN with the same number of hidden layers.

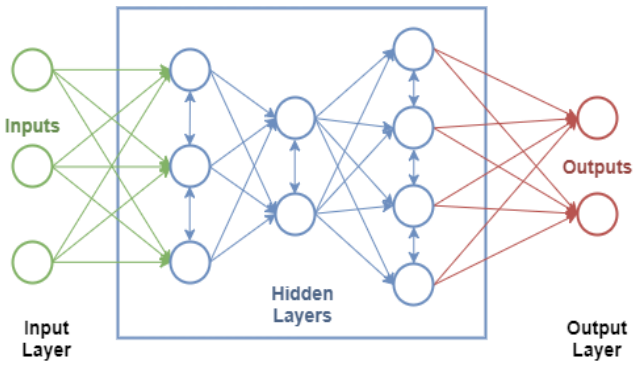


Figure 2. Architecture of a recurrent neural network

Typically, a FFNN remembers what it learned in training only and has no inherent notion of time; it is a major short coming of traditional neural networks. RNNs are able to overcome this

issue by introducing loops which allow information to be passed from one stage to the next; thus, allowing important information to persist in the network. The chain like structure of this network immediately relates it to a sequence making it a natural and powerful architecture to implement on sequence data. RNNs are much better suited to dealing with sequences, context modelling and time dependencies due to the memory they possess.

FFNN are at times too constrained, they are only able to accept a fixed size vector as input and produce a fixed size vector as output. A RNN has the ability to operate over a sequence of vectors. It takes many input vectors and is able to process them before returning other output vectors; Figure 3 below loosely describes this. The architecture of the LSTM used for this research is a "many to one" style as there are a number of feature vectors per time step being fed into the network as a time-series. [36]

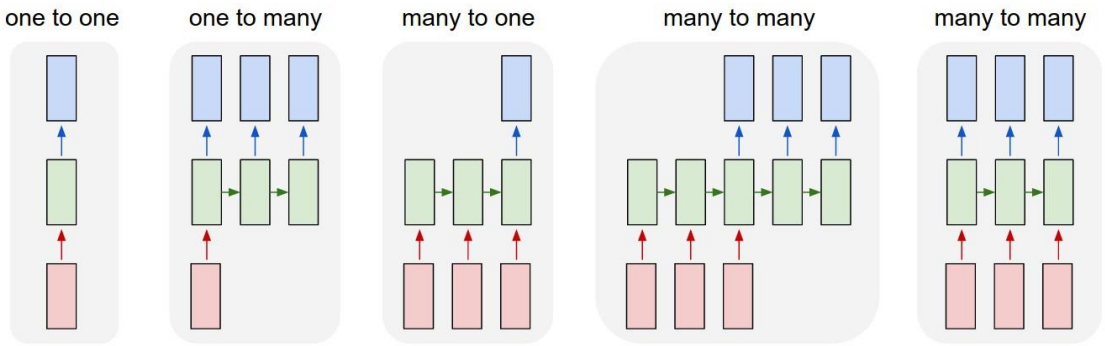


Figure 3. Input and Output architectures of a RNN, [36]

2.4.2 Long-Short Term Memory Network

Long short-term memory (LSTM) is a RNN architecture published in 1997 by Sepp Hochreiter and Jürgen Schmidhuber. LSTM units are also referred to as LSTM memory cells due to interpretation of gating units' likeness to computer memory.

RNN is a very powerful sequence model but it has been stated that it is difficult to train, whereas, the design and architecture of the LSTM overcomes this problem. The vanishing gradient problem found in the training of ANNs is a difficulty that occurs for gradient based learning methods and backpropagation. Neural networks weights are updated based upon the error function and the current weight found within a training iteration. [37]

Figure 4 below shows the architecture in an LSTM cell. The line that runs through the top of the cell is known as the cell state (C_t); running through the entire chain of LSTM cells. This is how information is passed between cells in the network.

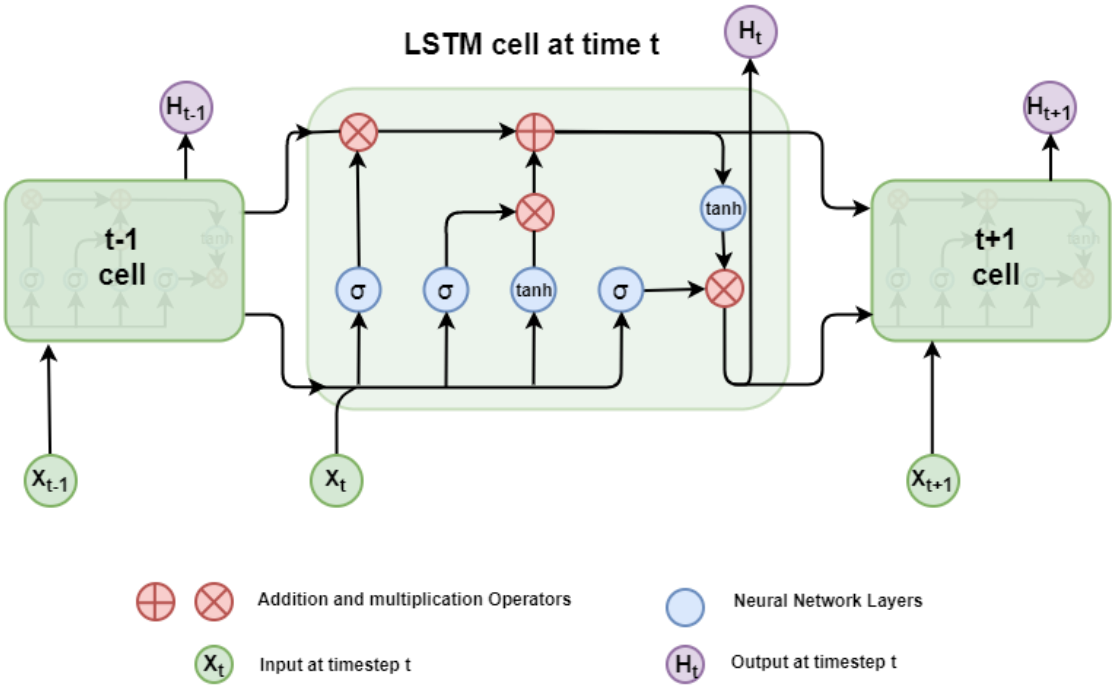


Figure 4. The architecture of an LSTM cell

Hochreiter and Schmidhuber defined this architecture to be constructed to "allow constant error flow through special, self-connected units"; the introduction of gated units allowed for this to be possible. Multiplicative input and output gates protect the memory contents from perturbation. The input gate shields alteration from irrelevant units and likewise the output gate unit protects other units from perturbations made by current stored irrelevant memory contents. [38]

The first decision made by the network is what information will be discarded from the cell's state; this is done using the "forget" gate layer (f_t) which is a sigmoid layer (σ). The current input (X_t) and previous output (H_{t-1}) are taken into consideration for this decision. The objective of the forget gate is to produce a binary decision for each number in the cell state; 1 denotes keeping this information and 0 votes to discard.

The information stored within that cell is determined in two parts. An activation function (the next sigmoid layer, σ) is applied as the input gate (i_t) where the result is sent to a multiplicative operator. The inner activation value of the previous time step is also passed through a multiplicative operator and summed with the recurrent self-connection. The hyperbolic tan layer creates a vector of candidate solutions ($\hat{C}_t$) that could be added to the cell's state.

The cell's state needs to be updated from C_{t-1} to C_t using all of the information obtained by these previous steps. The mathematical representation of this decision is:

$$C_t = f_t * C_{t-1} + i_t * \hat{C}_t \tag{2.1}$$

The result of the cell state is pushed through a hyperbolic tan layer which is "squashed" by a logistic sigmoid function. This sigmoid layer will decide what parts of the cell's state is the output (H_t). The various multiplication factors that regulate the scaling operations are controlled by the gated units (input, forget and output gates). [39] [40]

In conclusion, the LSTM is an extremely powerful implementation of a neural network that solves the vanishing gradient problem and if used correctly has the ability to produce highly accurate results on sequence based data.

2.5 Methods for evaluating a machine learning classifier

The purpose of training a classifier is to test on unseen samples of data, otherwise the model could easily memorise its inputs. This means that it would erroneously produce high classification scores, the classifier would not have learned to generalise correctly to cope with new data. This section will discuss some of the essential techniques for evaluating a machine learning classifier such as; a confusion matrix, accuracy rate, precision, recall and F1 score. The confusion matrix is presented first as the other evaluation methods stem from its description.

2.5.1 Confusion Matrix

A confusion matrix is employed to present the outcomes that were obtained by the classifier. This visualisation technique can be used in binary or multi-class classification. It will display the percentage of data the model has correctly classified versus the percentage misclassified showing a breakdown for each specific class. The columns represent the class that the model has predicted, whilst the rows represent the true class of the model. The diagonal of the matrix will represent the percentage of instances where the classifier has agreed with the truth of the dataset. A value of zero indicates no observations being assigned to this class, this is the aim for the off-diagonal elements.

The diagram below represents the structure of a 2 x 2 confusion matrix. The principles outlined here will extend to multi-class n x n confusion matrix, where n represents the number of classes.

		Predicted	
		Class 1	*Class 2*
True	*Class 1*	TP	FN
	Class 2	FP	TN

The correct entries are the True Positive (TP) and True Negative (TN), and the corresponding False Positive (FP) and False Negative (FN) entries have not been classified correctly.

2.5.2 Accuracy Rate

The overall correctness of a classifier is judged by the accuracy rate. The equation presented below represents the accuracy rate and is deduced from the confusion matrix above.

$$Accuracy\_Rate = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.2}$$

It is the percentage of correctly classified instances made by the classification model and is the simplest accuracy measure available for gauging the performance of a classifier. A higher accuracy rate obviously implies a better classification. It is good practice to examine the frequencies of the specific category labels in the test data, as a label could have 20 occurrences. If 18 of them are correctly classified then the 90% accuracy rate is not particularly remarkable. However, if the label has 100 instances and correctly classifies 90; this accuracy is a much better outcome.

Whilst the accuracy rate does provide insight into the percentage that is correctly classified, it is not enough to assess the performance of the model. For example, if there are 100 data samples to be classified and it correctly predicts 80 of them, then this would give an accuracy rate of 80%. At first this may seem like a good accuracy rate. However, if one class was walking and had 80 occurrences and the second class was jogging which had 20; the classifier could in theory have predicted the entire walking sample correctly and jogging incorrectly. Therefore, the accuracy rate of 80% would not be particularly good. This highlights the importance of examining a confusion matrix to improve the accuracy of a model.

2.5.3 Precision and Recall

As previously stated, accuracy is not the only metric required for evaluating performance. Precision and Recall are two metrics which provide greater insight into the true effectiveness of a classifier. They catalogue two different types of error that can be made during motion sensor classification.

Precision is the exactness of a classifier. It is the fraction of a correctly identified motion and the sum of identified instances for that motion. A higher precision value will denote fewer false positives within classifications, whilst a lower precision suggests more false positives. The leftmost column of the confusion matrix gives the values for precision. The precision is evidently the number of correctly classified TP within that predicted class.

$$Precision = \frac{TP}{TP + FP} \tag{2.3}$$

Recall is the sensitivity of the classifier; it is the ratio between the number of correctly classified observations and the number of true observations in the data. Higher recall suggests fewer false negatives and conversely a lower recall more false negatives. In the confusion matrix above, this is given by the top row. The recall is simply the ability of the classifier to identify all the positive samples.

$$Recall = \frac{TP}{TP + FN} \tag{2.4}$$

As the sample space grows it becomes increasingly difficult to increase the precision, especially when recall improves.

2.5.4 F1_Score

The last accuracy measure that will be used is the F1 Score (also known as F-measure). It is a measure of a test's accuracy. It can be considered as the weighted harmonic mean of Precision and Recall and is calculated as follows:

$$F1 = (1 + \beta) * \frac{Precision * Recall}{\beta * Precision + Recall} \tag{2.5}$$

Recall is weighed more than precision by a factor of β, if β=1 then recall and precision are considered equally important. When the value of β is less than one, then intuitively precision will be weighted higher than recall. The F1 score reaches its best value at one and worst at zero.

2.6 Programming Language and Toolkit Used

There are a number of programming languages and associated technologies that are suitable to implement for the motion sensor classifier. MATLAB, Python and R were investigated to find the most appropriate choice. Python was the programming language chosen for developing and testing with motion sensor classifications.

Presently, Python has many built in packages and libraries available for use such as: NumPy and SciPy for mathematical and scientific computing purposes respectively, matplotlib to create clear and concise data visualisations and Scikit-learn for machine learning purposes in Python. In addition, the (now very extensive) Pandas library provides two additional data containers (Series & DataFrame) for adding structure to data by annotating in the format of

named and indexed rows and columns. This functionality makes it useable and easily accessible for data analysis. [41]

The SciPy library contains a Signal toolbox [42] which was employed for the various signal processing tasks required and the Fourier transform. MATLAB [43] is a very popular option for signal processing; it is very useful for building preliminary models but not for achieving deployable solutions. Its Big Data capabilities and run-time speed are lacking in comparison to Python and is also a very expensive piece of software. [44] While this is available for use as a student, GRN would have to absorb the licence costs to continue using it after this work was complete. The SciPy alternative is inspired by the success of MATLAB's signal processing functionality. SciPy offers the same benefits and capabilities without the financial constraints for the business.

R [45] is a more statistically oriented programming language and environment. One of the main strengths of R is the ease of which a user can produce a high quality graphic, arguably superior to matplotlib for Python. The speed of R is particularly slow and functions often need to be run in parallel to speed up computation. However, implementation in Python is much easier, with high quality results available in shorter timescales. Whilst it is important to consider which is better suited for the tasks, when dealing with a considerable volume of data computation speed is an important aspect to consider when selecting a language. This is arguably the biggest downfall of R and one of the reasons Python was selected. However, the geosphere package in R was used for different aspects of handling and visualising the GPS data. [46]

Python 3.5.2 [47] was chosen after investigating various programming languages that would be suitable for this task. It was installed within Anaconda 4.0.2 (64-bit) [48], since it has an extensive list of analytical libraries. The Jupyter Notebook [49] was used due to the aesthetic of the computational environment it provides by combining code execution with rich text, mathematics and plots. The de-bugging functionality available also made it easy to work with Python code and data. Python is a general purpose programming language and has often been described as "easy to learn for beginners". [50] The more straightforward syntax makes the code human readable and enables a user to write programs at a faster speed. Python has the ability to successfully glue together large software components making it easy to integrate with every aspect of workflow.

The aforementioned Python libraries were utilised in combination with Tensorflow 1.2.0 for the purposes of this work. TensorFlow [51] is an open source software library for Machine Intelligence developed by Google. It allows for fast iterations of machine learning models and has an abundance of useful built-in functions. Tensorflow has a large and growing online

community meaning a large amount of code and models are available compared to competitors such as, Theano and Keras. It is optimised for big models and compile time is efficient. Tensorflow presented itself as the most favourable library for training an LSTM.

Lastly, Amazon Web Services (AWS) [52] was adopted to improve the run time speed of the network. AWS is a subsidiary of Amazon that provides on-demand cloud computing platforms for a user. The technology makes a virtual cluster of computers available through a paid subscription basis. Vast amounts of computing power are available which can be used to process and handle massive volumes of data quickly.

The Deep Learning AMI, Amazon Linux Version p2.xlarge [53] was installed to use in combination with the Tensorflow framework. It runs on the Amazon Elastic Compute Cloud (EC2) to provide a reliable, secure, and high performance execution environment for deep learning applications. It includes Anaconda Data Science Platform for versions of Python 3 and packages that facilitate easy integration with AWS, including launch configuration tools and many popular AWS libraries and tools.

3 Design and Approach

This aim of this section is to provide a clear and concise overview of the motion sensor classification development. It will begin by briefly discussing the data set used for the classifier, which was collected specifically for the purposes of this work. Each phase of the development process will be presented as a guide for the following chapters.

3.1 Data set

GRN had previously collected data from trials at Paris and London worn by the Scottish Rugby Union players. However, there was no information about what this data represented and it was not suitable to implement for this specific task. To obtain a dataset reflecting the movements chosen for the classifier, six volunteers at the Global Rugby Network office participated in a data collection activity. Participants were all male aged between 23 and 35, with varying heights and weights. They were instructed to follow a protocol of activities at a normal and comfortable speed.

One of the main reasons that GRN is interested in commissioning this research is to improve the performance hardware i.e. the player tracking devices. Each participant wore two devices (a top and bottom device), which resulted in twelve devices being used to capture the data. This was done to provide a visual aid for identifying any points where the device has failed to accurately record a motion. The bottom device was placed in the correct position shown in Figure 9 in section 4 and the top device placed directly to the side of it. Throughout the routine the placement of the devices was checked to ensure they were remaining tightly in place. The idea behind this was to take the mean value between the two devices for each recorded timestamp. The hope was to have an accurate signal between the top and bottom device.

Environmental factors such as the temperature registering at $28^{\circ C}$ (which is abnormally high weather conditions for Glasgow) meant that the participants were not performing in the most comfortable environment. This contributed to fatigue and exhaustion levels greatly and the subjects would not have been able to perform to the best of their ability. A number of the participants also experienced the fit of the vest to be uncomfortable and restrictive which contributed to exhaustion and the ability they were able to perform at.

Table 2 below lists the different exercises that this research aims to classify. It presents details of how long each exercise was performed for in one sitting and the number of instances of this exercise. This produces a column denoting the sum value of time for each exercise. It was important to consider this when creating a routine of activities due to multi-class imbalance. Class imbalance is the first problem to consider when developing a classifier. [54] It refers to

some classes in the data being highly represented whilst other classes are extremely underrepresented in comparison. The skewed distribution of classes can make many machine learning algorithms less effective, especially when trying to classify the underrepresented minority classes. A common example of this is when trying to predict fraudulent cases within insurance claims. The recorded number of true fraudulent cases will be much lower than the non-fraudulent.

Label	Activity	Performed for	No of Instances	Total Time
1	Standing	30 seconds	12	360 seconds
2	Walking	30 seconds	16	480 seconds
3	Jogging	30 seconds	20	600 seconds
4	Cruising	30 seconds	20	600 seconds
5	Sprinting	10 seconds	16	160 seconds
6	Vertical Jump	15 seconds	6	90 seconds
7	Horizontal Jump	15 seconds	6	90 seconds
8	Passing ring clockwise	60 seconds	6	360 seconds
9	Passing ring anticlockwise	60 seconds	6	360 seconds
			Total	3100 seconds

Table 2. List of Motions to be classified including how long they were performed for individually and in total for the whole routine.

The exercises displayed above were incorporated into routines in an attempt to try and make the data seem less "artificial" and more like natural, un-prompted human behaviour. Table 3 below shows a break down on the activities. For example, the walk-sprint was performed five times in one phase which lasted 160 seconds; this activity phase was performed 4 times during the routine. This was also broken up in this way to simulate a match or training session to provide a more accurate representation of the data. It was also to test the transition of speed between different exercises for the accelerometer. Another attempt to preserve the signal was to aid fatigue levels by breaking up a sprint with a walk etc.

Activity Phase	Repetitions in one phase	Total Duration	Iterations
Walk → Sprint	5	160 seconds	4
Cruise → Jog	5	300 seconds	4
Vertical Jump → Stand → Horizontal Jump → Stand	3	270 seconds	2
Passing ring clockwise → Passing ring anti-clockwise	3	360 seconds	2
	Total	3100 seconds	

Table 3. List of Activities and how many times this was repeated in one standing as well as how long this lasted, and also the number of times the activity was repeated

A formal definition of these activities is given in Appendix A.

3.2 Methodology

The methodology was compiled based on the CRISP-DM architecture (cross industry standard process for data mining); a common framework used by practitioners to plan any data mining project. The six stages of this are business understanding, data understanding, data preparation, modelling, evaluation and deployment; the flow of this process is charted in Figure 5 below.

Figure 5. CRISP-DM Methodology

The aim of this research is to be used as a foundation for future work in this application for GRN. Tri-axial accelerometer and gyroscope signals will be used for developing a motion sensor classifier. This contributes the link between the business understanding of the task at hand, and data understanding, which involved research of what the data was capable of being used for. The classifier development consists of six stages: data collection, pre-processing, data segmentation, feature engineering, training, and testing.

The data collection portion of this project has been discussed above and this is the only data used for the classifier. Within the training dataset, breaks that were taken in between activities resulted in unwanted data as it did not reflect any aspect of motions to be classified. In order to eliminate unwanted values from the dataset, the data had to be timestamped. Once a process was set up for getting an accurate timestamp for the data, the data could be time-sliced. This involved using the notes taken for recordings and simply erasing the data from the breaks. The data was then labelled according to these notes for each activity.

The pre-processing and feature engineering justifications are presented in the Signal Processing chapter of this report. This includes using a median then a low-pass filter as the initial pre-processing stages. The two files per participant are then merged after the data cleaning to produce the most accurate signal possible. The files were merged within a function using the following line of code from the Pandas (pd) library.

```
pd.concat([top_df, bot_df]).groupby('TimeStamp').mean()
```

It makes use of the concatenate function whilst taking the mean at each time stamp between the same columns of each data frame. The next step was to time-slice the data; this involved taking out all of the data that is not meaningful; such as the breaks taken in between activities.

The data will then be put into a window format using sliding windows and a Fourier transform will be applied to each window to suitably prepare the data for the classifier.

It will then be split into a training and test data set and fed into a LSTM where various hyperparameters will be altered to find the model producing the best classifications. Figure 6 presents the process of the stages employed.

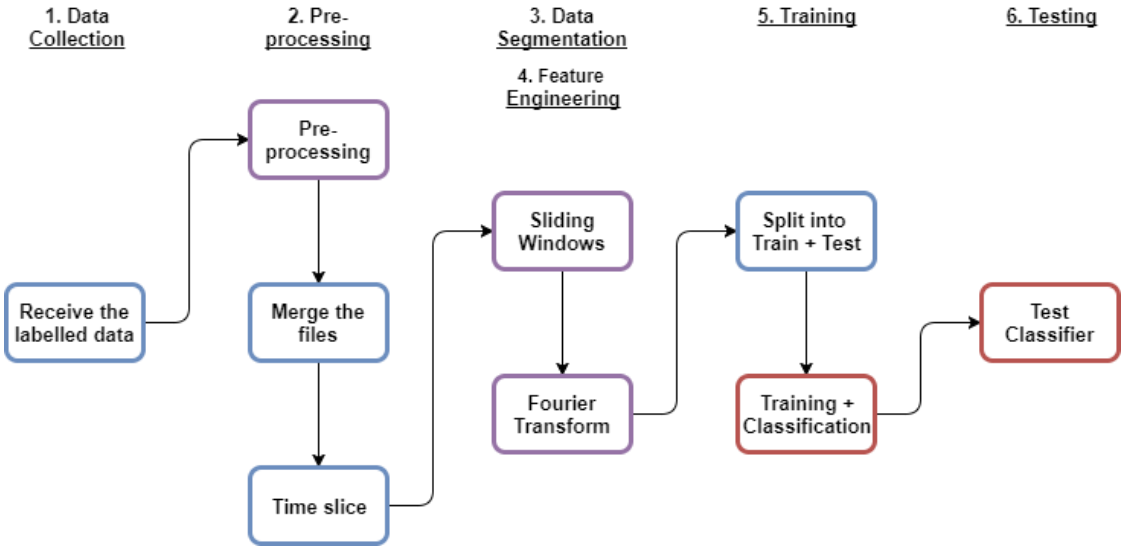


Figure 6. The Development Process

4 Data Acquisition

4.1 Player Monitoring Device

A Player Monitoring Device is used to capture the data, which includes a Global Positioning System (GPS) and Inertial Measurement Unit (IMU). A tri-axial accelerometer and gyroscope make up the IMU of this device. Figure 7 below shows the directions of the positive measurements of the device. The accelerometer will provide an insight into the dynamic acceleration force exerted by a player throughout the time the device is functional, whilst the gyroscope details the angular rate by measuring the rotations of the axes. The GPS is included for player tracking purposes to provide an enhanced set of statistics about the speed of a player and provide a visual of the movement on the pitch throughout the time operated.

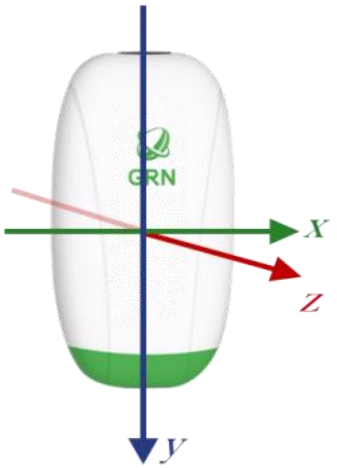


Figure 7. The GRN device showing the directions and measurements of the three axes

The accelerometer and gyroscope components have sensitivities; these tend to be high as they are intended to measure minute fluctuations in acceleration. The accelerometer component of the GRN device has a sensitivity measure of 16G (where 1G is the acceleration due to gravity at the surface of the Earth, equivalent of $9.8 ms^{-2}$). In order to derive a meaningful value from this and normalise the data, all raw accelerometer values need to be divided by 2048. This number is obtained by dividing the maximum value that the device can measure (32,768) by the sensitivity.

$$acclerometer\ conversion\ factor = \frac{32768}{16} = 2048 \tag{3.1}$$

The sensitivity of the gyroscope is 250 °/s which means that the raw gyroscope values need to be divided by 131, following the same process as before.

$$gyroscope\ conversion\ factor = \frac{32768}{250} \approx 131 \tag{3.2}$$

This is an important step for obtaining meaningful values of data. Without normalising the signal processing techniques the classification would not be as effective producing a poorer performance.

The rate at which a device samples is an important aspect to consider as it contributes to the accuracy of a movement. Most GPS watches and fitness trackers available to the general public sample at only one hertz (i.e. once a second); [12] for a professional athlete this would be highly inaccurate. An analyst needs to take into consideration the capabilities of a professional athlete and the speed they can move at in one second. The sampling rate of the IMU used by GRN is set to 100Hz, meaning the device captures 100 data points in one second whilst the GPS samples at 10Hz (producing 10 data points a second). For a motion processing algorithm IMU devices should be operating at a high sampling rate, typically it would be around 200 Hz. This is to provide more accurate results with low latency. [55]

The data generated by the accelerometer and gyroscope is a sequence of the different observations ordered by the offset (time), therefore making it time-series data. For each session that is recorded the device will produce a comma separated value file containing the accelerometer measurements and the offset, and similarly for the gyroscope. The file that corresponds to the GPS logging contains the offset, latitude and longitude (both in decimal degree format). Figure 8 below shows a snapshot of the raw accelerometer data. It consists of the first column as the offset and the second and fourth columns are the X, Y and Z measurements respectively. It shows ten rows of data, which amounts to one tenth of a second.

0	649	319	1930
10	666	326	1927
20	669	318	1914
30	678	317	1926
40	656	316	1936
50	647	319	1920
60	645	326	1930
70	658	319	1934
80	672	317	1950
90	679	324	1954

Figure 8. Snapshot of the raw accelerometer data from participant one's csv file

The data collection activity produced approximately 72 million data points from the accelerometer and gyroscope alone. After the files were merged (which cut the size directly in half) and the data was truncated to remove un-meaningful data, there was approximately 17 million data points available to use for training the LSTM classifier.

4.2 Placement of the Tracker

Figure 9 below shows a mock-up of the vest employed by GRN to hold the device during a match or whilst training. The device is placed between the shoulder blades; precisely it will sit between the T2 to T6 vertebrae with the longest dimension being parallel to the spine. The pouch the device is held in must ensure it remains tightly in place so the device cannot rotate on any axis whilst operating. The fit of the vest contributes to this, it needs to be tight enough to hold the device securely in place, but not so tight that it is uncomfortable and restrictive to a player.

Figure 9. A mock-up of the GPS vest used by GRN to hold the device

Aside from the fact that choice of spine placement is to comply with the Rugby Union regulations set in place for player monitoring devices; between the shoulder blades has been deemed as the most favourable placement. Common positions for IMU include the waist and hips, for example the England rugby team had previously placed the devices within a pouch in their shorts. [56] However, the hips are exposed to a lot of left and right movement, and a lot of gyration which will cause the data to be less accurate and less consistent; even more so when in a high impact sport environment. [12] To summarise, placing the device between the shoulder blades will remove any superfluous movement that would cause the gyroscope to record inaccurate data.

5 Signal Processing

Human data from wearable sensor technology results in “noisy data”. Classical techniques for signal processing need to be employed before any machine learning operations to improve prediction accuracy. The most common relationship between the two is that signal processing is used as a pre-processing (data cleaning) step before machine learning applications (classification). It is important to consider digital signal processing (DSP) before evaluating the efficacy of a predictive model. This section will discuss the use of DSP in the motion sensor classification project and the reasoning behind the techniques chosen.

5.1 Signal Pre-processing

In any data-driven project, pre-processing is an extremely important and commonly overlooked step. It consists of performing multiple imputations, filtering data, replacing outliers and extracting/selecting features. It is also good practice to observe the distribution of the data. This enables a visualisation as to whether there are any minority values or outliers that can skew the distribution of the data. The techniques employed during pre-processing can help to improve the accuracy rate of a classifier. Figure 10. below shows the order of techniques used for pre-processing particular signals. The values for the filters were inspired from an existing study in human wearable classifiers, [57]. The outputs shown at the end of this diagram are the time domain signals that will be fed into the LSTM classifier.

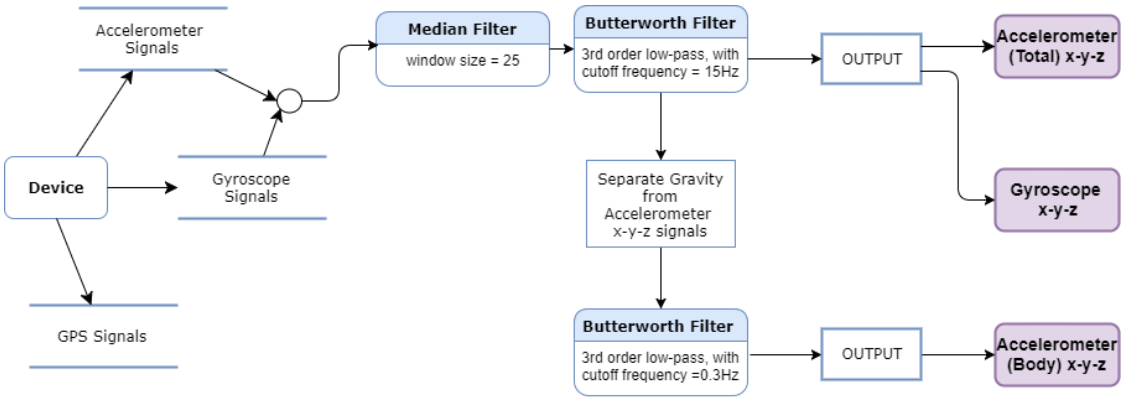


Figure 10. Filters applied at pre-processing stage of project

The GRN dataset will not have any missing values, so there is no need to consider any form of imputations. The first step for pre-processing the signal was to visualise it. Data visualisation is a general term that describes placing data in a visual context in an effort to aid the understanding of the significance of data. Visualisation allows for patterns, trends and clusters

of data points to be exposed and recognised by the human eye. A spike on a graph could be caused by the inaccurate logging of data by the tracker. Alternatively, it could be a rapid change caused by a change in movement or speed. Visualising the signal can provide an insight into what different patterns could be and effective methods to apply for signal processing.

The accelerometer X axis signals of the top and bottom device of participant one is shown in Figure 11. The difference between the two signals is highlighted, which shows outliers present in one signal that are not in the other. Taking the mean between an outlier and an accurate signal at a certain timestamp would not result in the most accurate signal possible. This illustrates the reasoning behind choosing to clean the signals before taking the average between them.

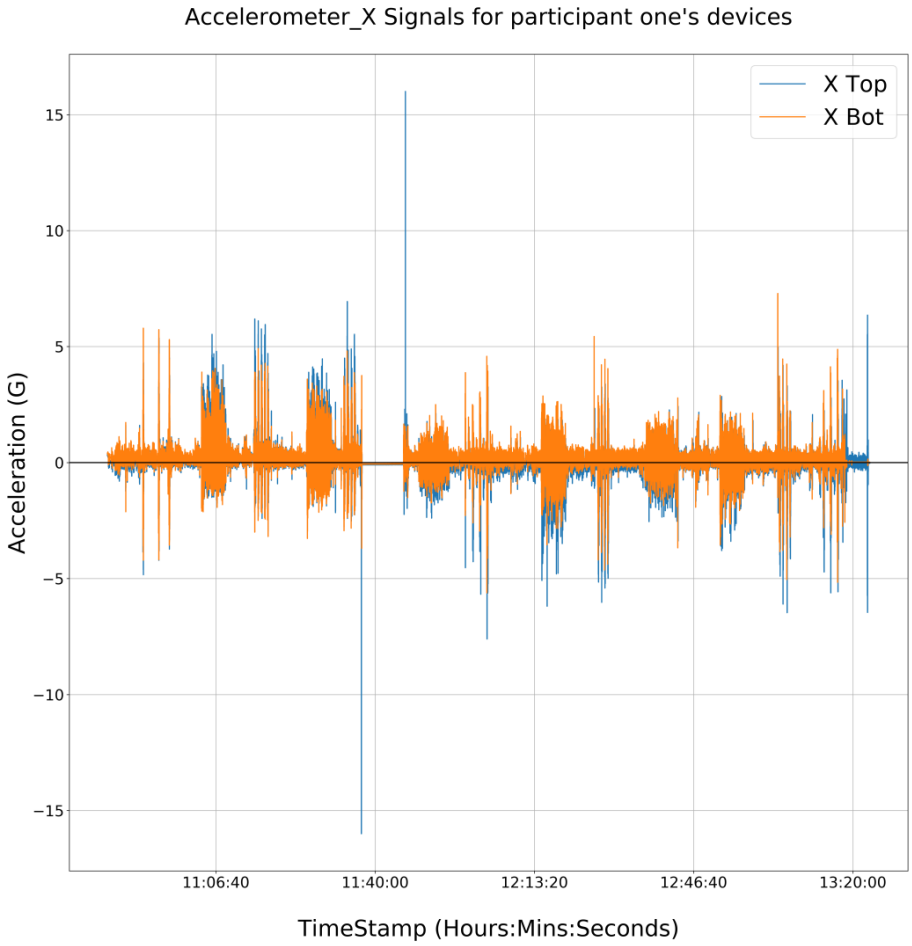


Figure 11. Raw Accelerometer X Signal for participant one showing the difference between the signal for the top and bottom device

5.1.1 Median Filter

Some form of random variation is always inherent in any data collected over a period of time; this variation is known as “noise”. A smoothing filter is used to take account of noise by reducing or eliminating the effect it has on the signal, thus allowing any important patterns to stand out. An accelerometer in particular is exposed to a number of high-frequency noise components so a smoothing filter is important for producing a more refined signal.

An investigation into a number of different smoothing filters appropriate for the IMU data was completed. From this, a moving average and median filter were directly compared to make an informed decision about which would produce the better outcome.

A median filter was chosen and administered to smooth the accelerometer and gyroscope signals. A window size of 25 was chosen to smooth for every quarter of a second. This was computed using the Signal toolbox available in the SciPy package: [58]

```
sp.signal.medfilt(data_frame[column], kernel_size = 25)
```

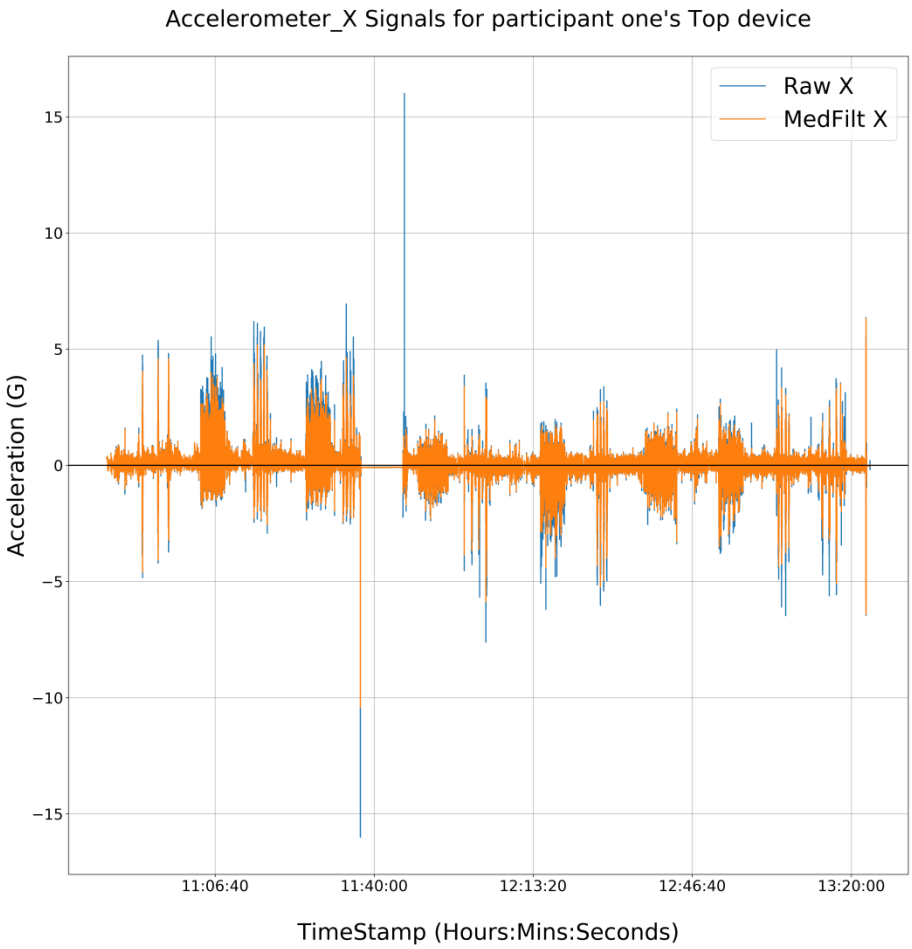


Figure 12. The accelerometer X axis signals showing the raw signal and the median filtered signal for participant one’s top device

Figure 12 above provides a visualisation on the performance of the median filter. It shows the original raw signal (blue) and the signal with the median filter applied to it (orange). It shows that the majority of unwanted outliers have been removed.

The transition between exercises can cause an algorithm to easily misclassify; using a smoothing filter should take account of this. The median filter was implemented to improve the result of the classifier; however one problem that is exposed is some short duration activities may not be classified correctly. [59]

5.1.2 Low-Pass Butterworth Filter

Low-Pass filters are used by supplying a cut-off frequency; they pass signals with a frequency lower than the cut-off value and attenuate signals with higher frequencies. When the noise components are isolated to a specific frequency range, a low-pass filter should be used to diminish it.

A 3rd order low pass Butterworth filter was selected to execute this. A cut-off value of 15Hz is sufficient to use for capturing human body motion as 99% of energy is contained below this threshold. Wearable technology employed for human motion can cause exorbitant noise to be recorded, which results in the appearance of outliers on the signal. This is the reason for employing two filtering techniques for removing noise. The role of the median filter was to simply smooth the signal; however some outliers were not accounted for. Using a low pass filter would remove any of these remaining inconsistencies resulting in a more accurate signal, such as the tracking device inaccurately recording the data.

This was also implemented making use of the SciPy signal processing library. The cut-off frequency supplied to the Butterworth filer needs to be normalised in the range of (0, 1). [59] This is calculated by making use of the Nyquist frequency; which is defined as half of the sampling rate. The following equation is used to normalise the cut-off frequency for the low pass Butterworth filter and shows the values used.

$$normal\_cutoff = \frac{cutoff\_frequency}{nyquist\_frequency} = \frac{15.0}{\left(\frac{100.0}{2}\right)} = \frac{15.0}{50.0} \tag{5.1}$$

This is used to build the Butterworth filter which gives two outputs; the numerator coefficient and the denominator coefficient of the filter. A forward-backward filter in the signal processing toolbox is used to apply the filter. The function applies the low pass filter twice, once forward and once backwards as the name suggests by using the coefficients of the filter. [61]

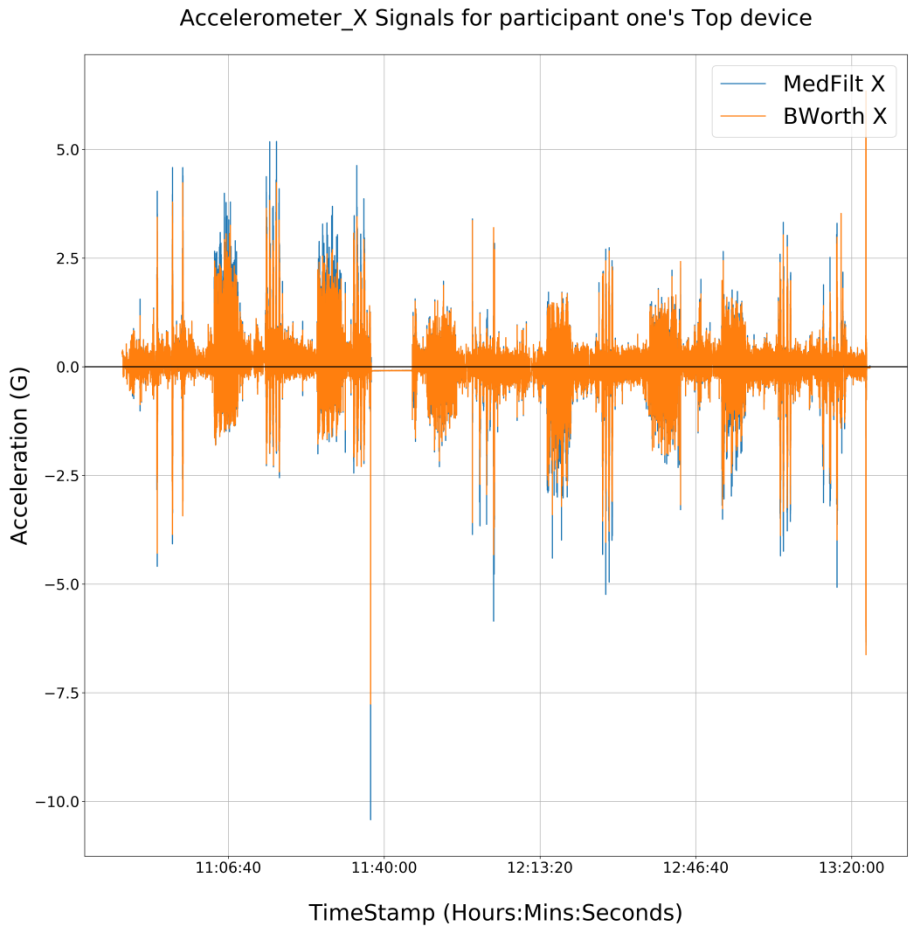


Figure 13. The accelerometer X axis signals showing the median filtered signal and the 3rd order low-pass Butterworth filtered signal for participant one's top device

Figure 13 shows the result of the 3rd order low pass Butterworth filter compared against the median filter. The remaining unwanted spikes have been removed to produce a more refined signal. Figure 14 shows the total amount of noise and outliers that have been removed from the signal. Signal in Appendix A shows a graph of the raw signal before any pre-processing against the end result after all the data cleaning steps have been applied.

The functions for the Butterworth filter are supplied for reference in the appendix. After this stage was carried out, the two files that were produced for each player were merged as referred to in section 3.2. After this the data was "time-sliced" (truncated) which removed the unwanted parts of the data, i.e. when a motion that was to be classified wasn't being performed. This explains some of the unwanted spikes that remain, as they were not meaningful parts of the data. Also, the point on the graph where the signal is zero is from the vests holding the trackers being temporarily removed.

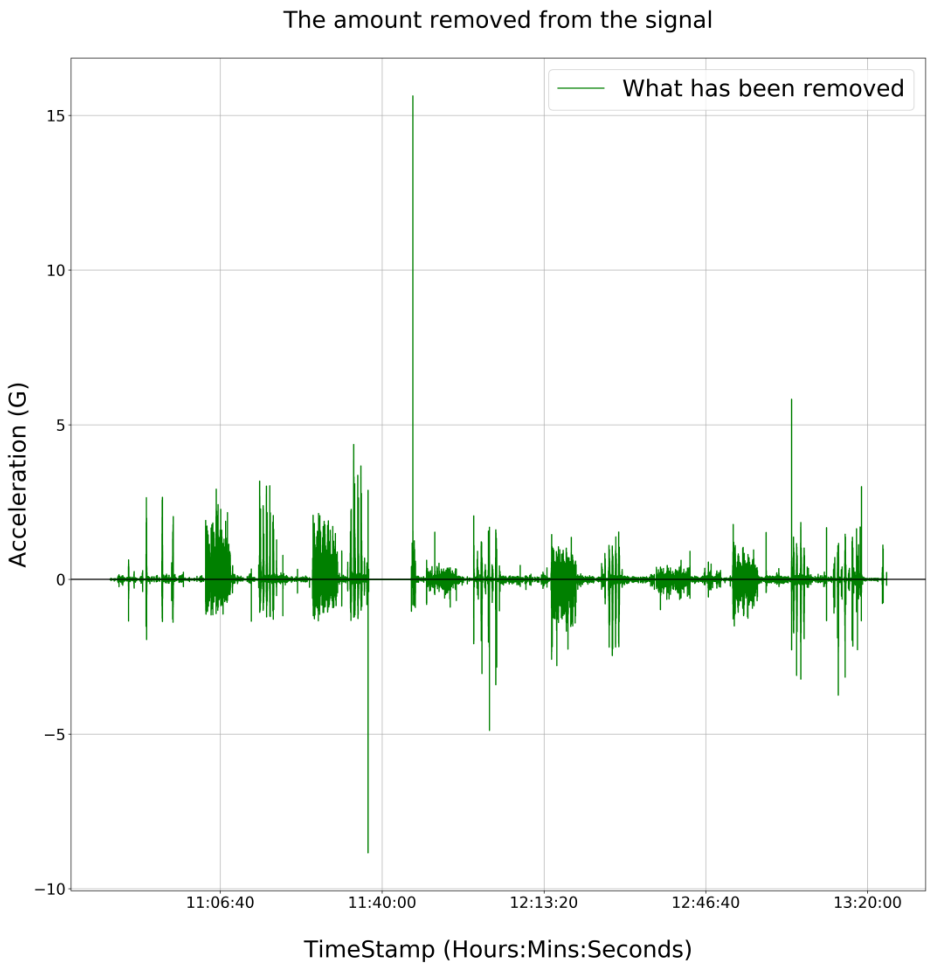


Figure 14. The noise and outliers removed from participant one's top device, accelerometer X signal

5.1.3 Accelerometer Signals

A raw accelerometer signal is comprised of three basic components; movement, noise and gravity. For the purposes of the classifier, only the movement component is of interest. As discussed above the noise has already been filtered out, the next step was to account for the gravity and remove it from the signals. Separation of these signals can become increasingly problematic, especially when investigating any rotational based movements.

An accelerometer is exposed to a great deal of gravity during the time it is operational. A low-pass filter could be used again to distinguish the two remaining portions of the signal. The result of the low pass filter was deemed to be the gravitational element of the signal by using a cut-off frequency of 0.3 Hz. This value has been observed to be the optimal corner frequency for constant gravity signal, as the assumption drawn is that gravitational force has only low frequency values. By subtracting this result from the accelerometer signal, only the body por-

tion of the signal remained. The Python code used to implement this is provided in Figure 15 to provide a clear explanation of this process.

```
def separate_accelerometer_from_gravity(accelerometer_data):

    """ The function used to separate the gravitational component
        from the accelerometer signal. 3rd order low-pass
        Butterworth filter with cutoff frequency = 0.3 Hz is
        used to identify the gravity.
    """

    sampling_rate = 100.0
    nyquist_freq = sampling_rate / 2.0

    body_signals = []
    for channel in accelerometer_data:

        # Low-Pass filter for 0.3 Hz to split gravity component
        gravity = butter_lowpass_filter(accelerometer_data[channel],
                cutoff_freq = 0.3, nyquist_freq, order=3)

        body = accelerometer_data[channel]
        body -= gravity
        # Assume that body signals have lost gravity component

        body_signals.append(body)

    body_accel_signals = np.array(body_signals)

    return body_accel_signals
```

Figure 15. Python code for the function implemented to separate the gravity and body components of the accelerometer signals

5.2 Time-Series Segmentation

Time-series data is often represented as a sequence of discrete segments of finite length. In time-series analysis, segmentation is a method that involves partitioning an input time-series into a sequence of discrete segments. It is important to consider how to represent time by converting the numeric points into a meaningful representation. A sliding windows algorithm was the approach chosen for this. This type of "windowing" technique is implemented to reveal the underlying properties of a signal and improve continuity representation.

The sliding windows algorithm implemented involved the time-domain signals being sampled in fixed-width sliding windows of 2.56 seconds and with 50% overlap between the windows. This reasoning was taken from [57] the same paper that inspired the values of the filters due to the success of the experiment. The value of 2.56 seconds was chosen because:

- The cadence of an average person walking is within the range of [90, 130] steps/min.
- For classification tasks, it is preferential to have one complete cycle of a motion in a window, e.g. two steps is one complete walking cycle.

This means that there will be 256 samples in each window, this is obtained from 2.56sec × 100Hz = 256 samples. The label corresponds to the sample of data in each discrete time interval. Figure 16 shows a representation of the data after the sliding windows algorithm has been applied. This method of windowing technique has shown success in the past in [28] and also in [27] that achieved a prediction accuracy of 93.39%.

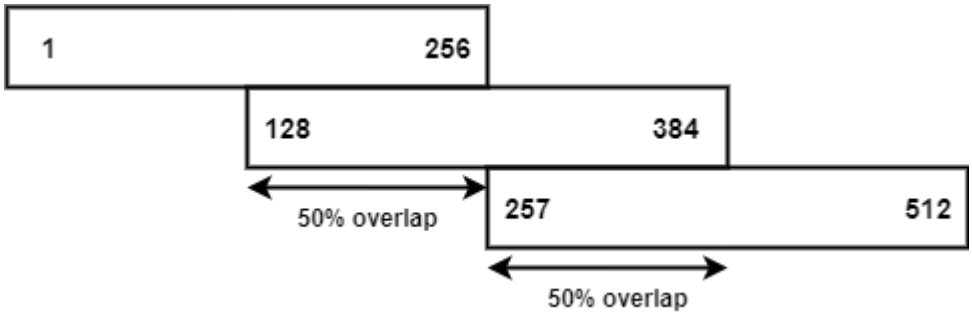


Figure 16. A representation of the data after applying the sliding windows algorithm

5.3 Feature Engineering

Table 4 below shows the time and frequency domain signals that were obtained for use in the LSTM classifier. The methods to obtain these signals are described below.

Acc_total_Signal	Gyro _Signal	Acc_body_Signal	Euclidean Magnitude
Time_x	Time_x	Time_x	Acc_mag
Time_y	Time_y	Time_y	Gyro_mag
Time_z	Time_z	Time_z	Acc_body_mag
Fourrier_x	Fourrier_x	Fourrier_x	
Fourrier_y	Fourrier_y	Fourrier_y	
Fourrier_z	Fourrier_z	Fourrier_z	

Table 4. Time and Frequency domain features

5.3.1 Fourier Transform

The Fast Fourier Transform (FFT) algorithm was used as a method for extracting a set of features from the signals. The algorithm computes the discrete Fourier transform (DFT) of the sequence supplied to it. The purpose is to convert a time-domain series to a frequency domain. The DFT is defined by the following formula:

$$X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-i2\pi kn}{N}} \qquad k = 0,1,\ldots,N \qquad (5.1)$$

This produces values of complex types with a real and imaginary part (a + bi). These numbers are useless in this situation, to overcome this; the power log spectrum is computed. Depending upon the data, the FFT could reveal any harmonic content of the signal such as cycles and repetitions; these will be visible by a few prominent peaks on when plotted.

The signals are mapped into the frequency domain through each window created by the sliding window function. The work in [28] had great success for classifications by using this method.

There are built in functions to compute the FFT algorithm available in Python through the NumPy and SciPy libraries. SciPy has a slightly faster computation time, however; it requires an extra stage to convert to complex output. NumPy may take longer to compute but the output is ready to use straight away. [63] For this reason, NumPy's fast Fourier transform module was used to transform the time-domain signals to a frequency domain signal.

5.3.2 Euclidean Magnitude

The Euclidean Magnitude (also commonly referred to as the norm) was used to supply extra features to the classifier. This stage involved taking the magnitude across the x, y and z axis of each of the accelerometer, gyroscope and accelerometer body signals. The formula used for this was:

$$\|m\| = \sqrt{x^2 + y^2 + z^2} \tag{5.2}$$

In Python, this made use of another NumPy module called linear algebra. [64] This module had a function that would take the norm across the three axes, returning a single value for each timestamp.

6 Neural Network Design

This section will discuss various aspects of the neural network design, including a number of decisions made/trialled to improve the performance of the algorithm.

6.1 Cost Function

In any data mining project a model which minimises the sum of squared residuals can be defined as the "best" model. In this instance, the residual of a data point is defined as the difference between the true (observed) class and the class predicted by the model. The concept of minimising a function is a building block of supervised learning, and this function is known as the Cost Function (also commonly known as the Loss Function). By finding the global minimum of a cost function, the cost associated with the model is minimised.

Traditionally, a cost function is minimised through a learning process known as gradient descent. How well a cost function has performed with respect to its training sample may depend on different weights and biases.

6.1.1 Learning Method

An optimisation algorithm for the motion sensor classifier could mean the difference between good results in minutes or in hours. Adam is an optimisation algorithm that can be implemented instead of the classical stochastic gradient descent approach to update network weights. Gradient Descent is not the most efficient learning method to use for training any Deep Neural Network. The Adam optimiser was first presented in [65] by D. Kingma and J. Ba, who aimed the method towards toward machine learning applications with large datasets. The effectiveness of Adam compared to previous techniques is highlighted in Figure 17 below. It is apparent that Adam makes faster progress in terms of time and number of iterations. The benefits of implementing the Adam optimiser highlighted by the authors are the ease of which it can be implemented, it is computationally efficient and it requires very little memory. Hyper-parameters have intuitive interpretation in the optimiser so typically require little or less tuning. The Tensorflow library includes a built-in function for implementing an Adam optimiser.

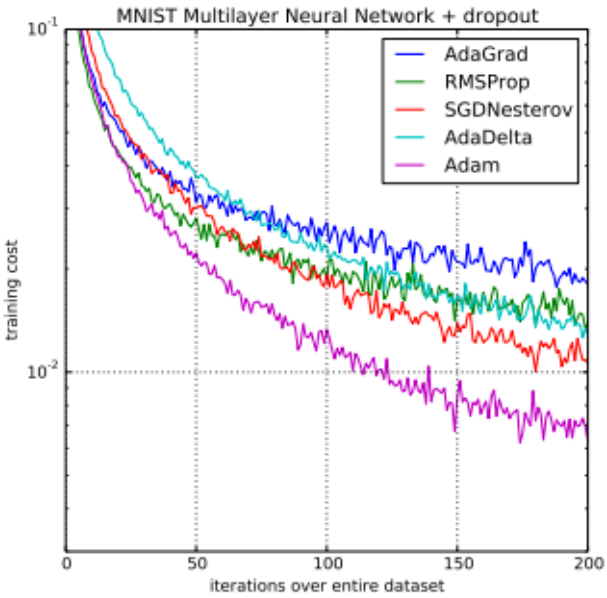


Figure 17. The results of the Adam Optimiser compared to classical techniques on training a multilayer neural network, from [65]

6.1.2 Training

Training a neural network can be described as a process for finding a set of weights and biases for computed outcomes to closely match the known outcomes of the training data instances. When a set of values for weights and biases that produce reliable results has been uncovered, the neural network should have the functionality to make predictions on unseen data that does not contain labelled outcomes. [37]

One approach to use for training a neural network is batch training. Batch training involves working with the entire dataset during training. A batch is the number of data instances used in an iteration of training. It involves learning over a group of patterns (multiple data samples) as opposed to learning with one data sample at a time (online learning). The weights and biases are only updated after all the inputs have been presented to the neural network when batch training is being used.

In classification related machine learning tasks, a loss function represents the "price paid" for predictions that are inaccurate. This LSTM network uses an L2 Softmax loss function. It is the final layer of the neural network which yields the given performance scores for each class in the data.

6.2 One Hot Encoding

Categorical data can often be difficult for machine learning algorithms to handle as they require numerical data as both inputs and outputs. This problem is easily overcome by transforming the data in two steps; integer encoding and one hot encoding. A nominal variable has no sense of ordinal relationship whereas an integer variable possesses this quality. When integer encoding does not provide enough understanding for a classifier to harness this relationship, one hot encoding can be applied. This type of encoding allows an algorithm to assume an inherent, natural order of categorical classes resulting in better performance accuracy. [66]

The labels for the data, which are displayed in Table 2 of Section 3.1, were already supplied in an integer format to avoid an extra transformation to the data. For the motion sensor classifier, one hot encoding was applied directly to the integer representation of the categorical classes. It involves removing the inter representation and replacing it with a binary string that is unique to each integer category label. A representation of one hot encoding for four unique integer labels is presented in Table 5.

		One Hot Encoding			
	1	1	0	0	0
Label	**2**	0	1	0	0
	3	0	0	1	0
	4	0	0	0	1

Table 5. One hot encoding for four unique integers

6.3 Hyperparameters

Hyperparameters can have a significant impact on the performance and results of a neural network. Any machine learning model needs to learn the parameters that give produce the optimal mapping from inputs to outputs. The cost of not fine-tuning hyperparameters in the learning process can be high. A "grid search" approach is the most traditional and prevalent technique used for hyperparameter optimisation and was employed in this project. It is a process of exhaustive enumeration of trying different combinations of hyperparameters in the algorithm. It can be argued that it is a costly method in terms of time however, for an exploratory research project provides insight into the effects of different combinations and should be helpful in later work. The hyperparameters that are to be tuned during training are as follows.

Hidden Layers – this is the only hyperparameter for the model itself (the remaining are all training hyperparameters). Random selection of these values was adopted to find the optimal

number to use and values between 20 and 35 are reported on. The optimal number for this hyperparameter allows the neural network to fit the training data well and learn to generalise.

Learning Rate – The learning rate determines the "step size" that is taken during the learning process. The values for the learning rate can be in the range of [0.0001, 0.1]. A small enough step size is required for a model to learn to generalise correctly. The learning rate needs to be optimised to a suitable value as, too high a value will not learn properly. Similarly, if the value is too low the model will not learn properly; it can cause training to get stuck in local minima or take too long. A complete explanation of the learning process and rate is available in [67]. The values investigated were 0.001, 0.0015, 0.002 and 0.0025. They were all sufficiently small enough for the classifier to learn during the training session.

Batch Size – the batch size used can have an impact on the model. A smaller batch size allows more updates to be made however a larger batch size allows the updates to be computed more efficiently. The batch size can be optimised separately from the rest of the hyperparameters; it was set to be 1500 in this experiment. Other batch sizes (both higher and lower) caused the training process to drastically slow down or the resulting classification was much poorer.

Training Iterations – this is the number of times that the algorithm loops over the dataset. For this experiment the algorithm looped over the dataset 300 times. It was treated in a similar fashion to the batch size; other values caused the training session time to increase and did not improve the classifier performance. Altering the training iterations can be a powerful method for preventing overfitting in the model. [68]

6.4 Data Split

When the objective of a data analytics project turns towards the development of a model used for classifying, there needs to be a means of assessing its accuracy, reliability and credibility. The final stage in preparing the data gathered for the classifier is to partition it into a Train dataset and a Test dataset. The essential role of the training dataset is to provide the data that generates the model. The classifier will learn from this data when the model is training and the test dataset is left untouched throughout this phase. The test dataset is employed after the training stage to evaluate the performance of the model on unseen data. The underlying assumptions of this split are such that:

- The data available fairly represents the real-world processes being modelled
- The real-world processes are expected to remain relatively stable over time. A well-constructed model built on previous data is expected to perform adequately on future data

Therefore, if these assumptions hold true, holding back some of the present data is a fair approximation to having future data for testing.

The data employed for the motion sensor classifier was split using a feature of the Scikit-learn library. The model selection submodule of Scikit-learn contains a feature for splitting into train and test data. [69] This function randomly partitions data the data before splitting. It allows the percentage of each dataset to be specified. The stratify parameter used within also allows for an even distribution of the classes between the train and test datasets. Appendix B shows the bar plots illustrating the split for each class between the training dataset and the test dataset. Figure 18 shows the flow of this process.

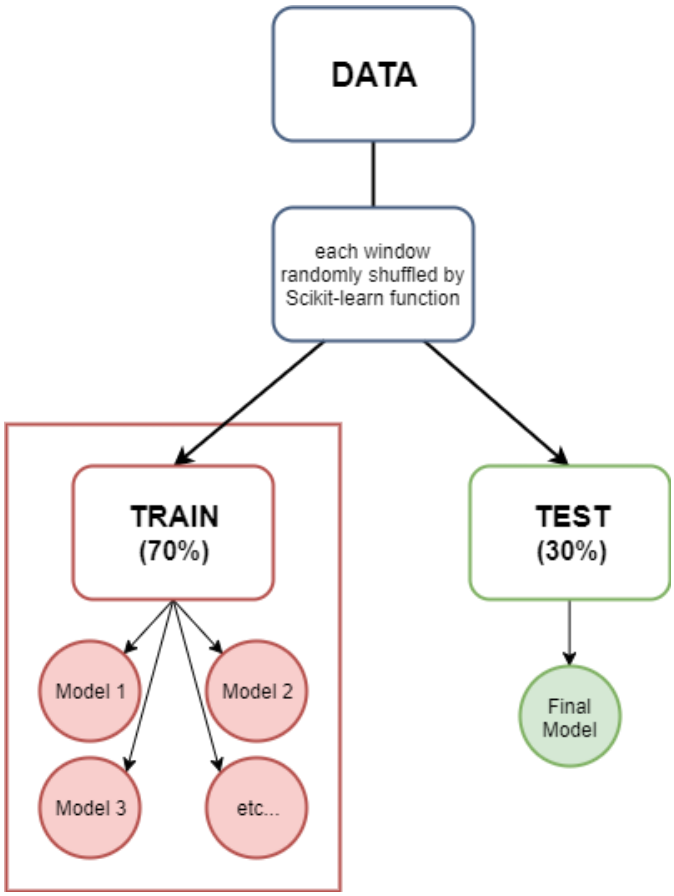


Figure 18. Data Split

7 Analysis & Results

This section will provide a summary of the results produced by the LSTM algorithm. Various configurations of hyperparameters and feature sets were fine-tuned on this algorithm. As previously stated, an approach of exhaustive enumeration was employed to find the optimal combination and observe any insights in the behaviour of the network

Table 6 below shows the combination of different feature sets that have been applied to these algorithms, where acc and gyro denote the accelerometer and gyroscope readings respectively. All of these signals (which are being treated as different combinations of feature sets) are time-domain signals that have been achieved by following the process laid out in chapter 5 *Signal Processing*. Fourier transform variations of this signal have also been trialled; a prefix of "f" is used when referring to the Fourier transform signal and "mag" is used when referring to the magnitude implementation of the signal.

	Feature Set	**Accelerometer Signals**	**Gyroscope Signals**
1	Original signals	acc_X, acc_Y, acc_Z	gyro_X, gyro_Y, gyro_Z
2	No gravity signals	body_X, body_Y, body_Z	gyro_X, gyro_Y, gyro_Z
3	All signals	acc_X, acc_Y, acc_Z, body_X, body_Y, body_Z	gyro_X, gyro_Y, gyro_Z
4	With magnitude signals	acc_X, acc_Y, acc_Z, acc_mag	gyro_X, gyro_Y, gyro_Z, gyro_mag
5	Accelerometer only signals	acc_X, acc_Y, acc_Z, body_X, body_Y, body_Z	
6	Fourier Transform signals	facc_x, facc_Y, facc_Z	fgyro_X, fgyro_Y, fgyro_Z

Table 6. The descriptions of each of the feature sets tested

7.1 Two Stacked LSTM Cells

The results below were all passed through a RNN with two stacked LSTM cells, the Python code for this algorithm is represented in Appendix C. Using stacked cells is what makes it a *deep* neural network.

7.1.1 Original Signals

The results of the first feature set are presented below in Table 7. The number of hidden layers was altered whilst the learning rate and lambda loss amount were kept constant at 0.0025 and 0.0015 respectively to find the optimal number of hidden layers.

Hidden Layers	Accuracy (%)	Precision (%)	Recall (%)	F1-Score	Batch Loss
20	69.291824	69.191069	69.291819	0.686105	0.960350
25	70.818067	70.457468	70.818071	0.704185	1.004126
28	73.382181	73.869218	73.382173	0.733548	0.946159
30	74.450552	74.952946	74.450549	0.743022	0.945834
31	75.122112	75.310360	75.122100	0. 747973	0.916978
32	77.564102	77.743261	77.564103	0.775422	0.900496
33	**80.402929**	**80.483431**	**80.402930**	**0.803301**	**0.861666**
34	76.159954	76.812028	76.159951	0.762359	0.982086
35	74.175829	74.688734	74.175824	0.739767	0.990814

Table 7. Results of fine tuning of the number of hidden layers on the first feature set with the learning rate kept equal to 0.0025

The configuration that gave the optimal result from this experiment was with 33 hidden layers, learning rate as 0.0025. The learning rate was fine-tuned keeping the number of hidden layers equal to 33, the results of this are displayed in Table 8 below.

Learning Rate	Accuracy (%)	Precision (%)	Recall (%)	F1-Score	Batch Loss
0.0010	71.398050	71.939312	71.398046	0.709904	1.187604
0.0015	75.000000	74.683727	75.000000	0.744932	0.957991
0.0020	74. 297929	74. 042799	74. 297924	0. 739059	0. 990942
0.0025	**80.402929**	**80.483431**	**80.402930**	**0.803301**	**0.861666**

Table 8. Results of fine tuning of the learning rate on the first feature set with hidden number of layers kept equal to 33

The optimal configuration was as previously stated with the number of hidden layers and learning rate equal to 33 and 0.0025 respectively, producing an accuracy measure of 80.40%. The precise confusion matrix for the optimal configuration is:

SPRINT	150	3	28	9	2	4	1
JOG	5	664	108	4	3	1	1
CRUISE	32	135	687	0	0	1	0
WALK	6	11	5	434	6	6	11
V JUMP	3	14	0	8	207	13	46
H JUMP	1	2	1	15	56	150	23
STAND	0	1	0	16	43	18	342
	SPRINT	**JOG**	**CRUISE**	**WALK**	**V JUMP**	**H JUMP**	**STAND**

Where the rows represent the true label and the columns are the predicted label. It is apparent that the sprint, walk and stand classes have all performed well in this algorithm. The LSTM is struggling to classify the difference between jog and cruise and a vertical and horizontal jump in comparison. The majority of data segments have been predicted correctly however there is a much clearer struggle between these than the aforementioned classes.

7.1.2 No gravity signals

The second feature set, was passed through the network in the same fashion and the results are displayed in Table 9. The learning rate and lambda loss amount were kept constant at 0.0025 and 0.0015 while the number of hidden layers was varied.

Hidden Layers	Accuracy (%)	Precision (%)	Recall (%)	F1-Score	Batch Loss
30	**66.880345**	**66.363483**	**66.880342**	**0.660842**	**1.188379**
31	64.896220	64.883212	64.896215	0.646942	1.238871
32	66.300368	66.680690	66.300366	0.661081	1.299446
33	66.483521	65.985293	66.483516	0.661145	1.266903
34	65.842497	65.868753	65.842491	0.657071	1.297687
35	65.445668	65.439890	65.445665	0.652127	1.294704

Table 9. Results of fine tuning the number of hidden layers on the second feature set

The best version of the model produced was with 30 hidden layers, it is also very apparent that this has not performed as well as the first feature set. The confusion matrix for this model is:

SPRINT	85	4	126	16	3	10	3
JOG	2	627	161	9	6	0	1
CRUISE	20	213	594	4	2	2	2
WALK	7	14	7	392	8	8	17
V JUMP	2	13	5	24	125	31	77
H JUMP	11	3	8	26	40	121	32
STAND	3	6	3	54	72	30	247
	SPRINT	**JOG**	**CRUISE**	**WALK**	**V JUMP**	**H JUMP**	**STAND**

Immediately, it is apparent that this is a much poorer result than those obtained for the original signals. The sprint class has not performed as well misclassifying as mostly the cruise. The two jump and stand classes have also had significantly poorer results than the original signals set. The walking class still classified well but was marginally poorer than the original signals result.

7.1.3 All signals

The third feature set was the accelerometer, gyroscope and accelerometer body signals. Table 10 denotes the various performance measures that were achieved during this run.

Hidden Layers	Accuracy (%)	Precision (%)	Recall (%)	F1-Score	Batch Loss
30	67.582422	67.533845	67.582418	0.671872	1.104750
31	69.230777	69.240812	69.230769	0.690421	1.100112
32	67.979240	68.099634	67.979243	0.680156	1.186175
33	69.993901	69.983077	69.993895	0.698022	1.093515
34	63.949943	63.489814	63.949939	0.636352	1.330015
35	**70.238107**	**70.353300**	**70.238095**	**0.698717**	**1.147799**

Table 10. Results of fine tuning the number of hidden layers on the third feature set

Enhanced results compared to the second feature set were achieved, however these results were still not as high-quality as the first (original) feature set. The optimal result found was with the number of hidden layers equal to 35 which gave 70.24%. The confusion matrix was as follows:

SPRINT	116	6	95	19	1	6	4
JOG	1	604	175	22	3	0	1
CRUISE	22	175	626	5	4	2	3
WALK	5	16	4	391	5	4	28
V JUMP	3	7	7	22	141	15	82
H JUMP	10	2	6	27	19	141	36
STAND	0	2	1	32	85	13	282
	SPRINT	**JOG**	**CRUISE**	**WALK**	**V JUMP**	**H JUMP**	**STAND**

Whilst these results are still not as good as the original signals feature set, it is obvious that it is much better than the no gravity signals. The sprint class has performed much better than previously, as has the stand class. The same issue with cruise and jog still arises and similarly with the jumps and stand classes. The walking class remains consistent.

7.1.4 Euclidean Magnitude Data

The outcomes of fine-tuning the number of hidden layers on the fourth feature set are exhibited in Table 11. This feature set included the Euclidean magnitude signals feature for the accelerometer and gyroscope in addition to the original signals.

Hidden Layers	Accuracy (%)	Precision (%)	Recall (%)	F1-Score	Batch Loss
31	67.521375	67.260709	67.521368	0.670008	1.134920
32	66.910863	66.633901	66.910867	0.665581	1.282758
33	**71.001226**	**71.699608**	**71.001221**	**0.702757**	**1.126809**
34	65.720397	65.909666	65.720391	0.654697	1.274866
35	67.857146	67.755131	67.857143	0.677217	1.163469

Table 11. Results of fine tuning the number of hidden layers of the fourth feature set

Similarly, this investigation did not produce better results than the first feature set. 71.00% was the best accuracy rate that the algorithm managed to reach. The confusion matrix for this feature set with the number of hidden layers set at 33 is presented below.

SPRINT	95	13	88	21	1	5	4
JOG	1	636	123	14	4	0	7
CRUISE	12	246	566	4	0	1	1
WALK	4	12	5	416	3	5	42
V JUMP	2	11	5	7	121	28	114
H JUMP	3	2	3	23	17	149	45
STAND	3	3	1	22	24	21	343
	SPRINT	**JOG**	**CRUISE**	**WALK**	**V JUMP**	**H JUMP**	**STAND**

The configuration for this network gave a marginally higher accuracy rate than the optimal configuration for the Euclidean magnitude signals (section 7.1.3). However, after investigating the different matrices the sprint class has performed much poorer despite the higher accuracy rate. Again, the walk class has performed well.

7.1.5 Accelerometer Data only

Feature set five was the accelerometer total and body only signals. It was selected to exclude the gyroscope signals as it consumes much more power than an accelerometer does. [70] Table 12 shows the full results of varying the number of hidden layers in this configuration.

Hidden Layers	Accuracy (%)	Precision (%)	Recall (%)	F1-Score	Batch Loss
30	78. 052503	78.091844	78.052503	0.776068	0.854365
31	**80.006105**	**79.903809**	**80.006105**	**0.798645**	**0.817508**
32	77.655685	77.802673	77.655678	0.773413	0.914106
33	78.052509	78.610008	78.052503	0.779254	0.850900
34	79.853487	80.035454	79.853480	0.796313	0.832503
35	78.205138	78.164623	78.205138	0.778327	0.861665

Table 12. Results of fine tuning the number of hidden layers on the fifth feature set

Surprisingly, this feature set produced an accuracy measure of approximately 80.00%; which is not a great difference compared to 80.40% from feature set one. The optimal result was with the number of hidden layers equal to 31, the confusion matrix for this was:

SPRINT	165	6	49	18	6	3	0
JOG	7	665	123	8	1	1	1
CRUISE	31	142	656	1	4	0	3
WALK	6	7	8	425	0	6	1
V JUMP	3	7	4	6	184	12	61
H JUMP	1	1	1	15	10	180	33
STAND	1	0	0	2	31	35	346
	SPRINT	**JOG**	**CRUISE**	**WALK**	**V JUMP**	**H JUMP**	**STAND**

Without the gyroscope data, the sprint class produced the highest number of correct classifications of all the feature sets tried so far. The same problems with cruise and jog and for the two jump classes and stand still arise. Walk is still consistently being correctly classified the most.

7.1.6 Fourier Transform signals

This feature set consisted of the Fourier transform versions of the accelerometer and gyroscope data. Table 13 below shows some results of configurations tried.

Hidden Layers	Accuracy (%)	Precision (%)	Recall (%)	F1-Score	Batch Loss
20	51.03785	51.227901	51.037851	0.503669	1.302989
25	51.678878	51.609079	51.678877	0.510935	1.295542
28	43.803426	45.555417	43.803419	0.420991	1.477041
33	49.572653	49.476274	49.572650	0.488366	1.351709
35	50.061053	50.552694	50.061050	0.491191	1.382882

Table 13. A highlight of the results of the FFT data

It is very apparent that these results are not as high as the previous signals that have been ran through the classifier. The FFT version of the other signals has also been attempted; however none of them obtained particularly good results. It is could be possible that the FFT is removing too much important and distinctive information from the signals.

7.2 Varying Window Size

To window size was varied from 256 to 128 (still with a 50% overlap) to investigate the networks performance on different activities. Table 14 below shows the results of different feature sets and hidden layers.

Feature Set	Hidden Layers	Accuracy (%)	Precision (%)	Recall (%)	F1-Score	Batch Loss
1	33	66.153139	66.074842	66.153142	0.659775	1.137405
2	30	63.804144	64.219447	63.804149	0.636294	1.273343
3	35	65.329468	65.719803	65.329469	0.652720	1.214749
5	**31**	**73.764485**	**74.198546**	**73.764491**	**0.733195**	**0.919023**

Table 14. Results of different feature sets and hidden layers with data window size = 128

This highlights that all of the metrics for evaluating the classifier have not performed as well as they did when the window size was 265. The best result obtained was with the accelerometer only data, the confusion matrix for this was:

SPRINT	212	27	112	35	5	9	4
JOG	11	1361	206	34	2	3	0
CRUISE	97	438	1181	11	4	9	3
WALK	10	21	14	880	0	18	4
V JUMP	5	23	9	7	282	78	168
H JUMP	14	2	0	52	23	304	112
STAND	3	2	1	5	25	114	616
	SPRINT	**JOG**	**CRUISE**	**WALK**	**V JUMP**	**H JUMP**	**STAND**

7.3 Analysis of the Optimal Configuration

The optimal configuration of the LSTM RNN was obtained using the accelerometer and gyroscope XYZ signals. The number of hidden layers was 33, the learning rate 0.0025, lambda loss amount 0.0015 and a batch size of 1500 looping over the dataset 300 times. The overall accuracy rate of this was 80.4%. Figure 19 below shows the confusion matrix normalised to the percentage of test data. The exact numbers of this confusion matrix are presented in section 7.1.1 above.

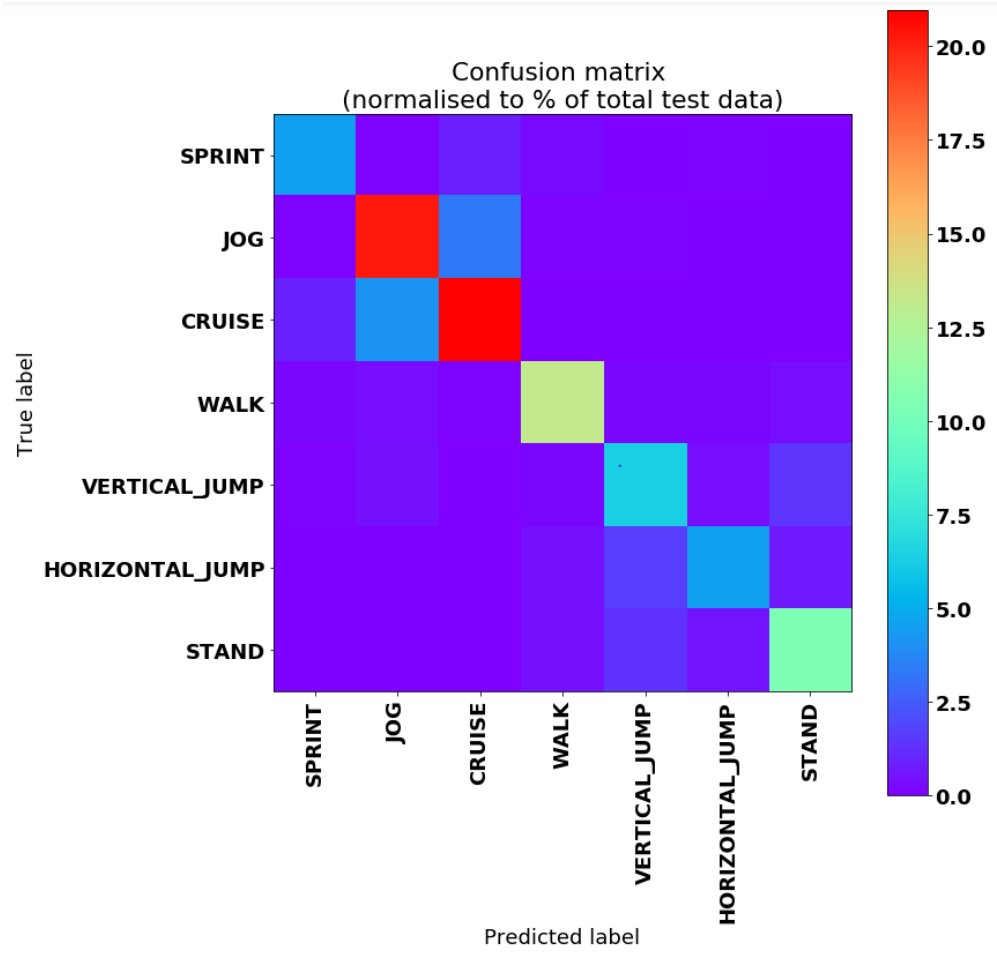


Figure 19. The confusion matrix for the optimal result obtained during training

The model predicted the sprint class well with few incorrectly classified instances, and similarly for walking. Figure 20 below shows the signals produced along the x, y and z axis for the sprint and walk classes.

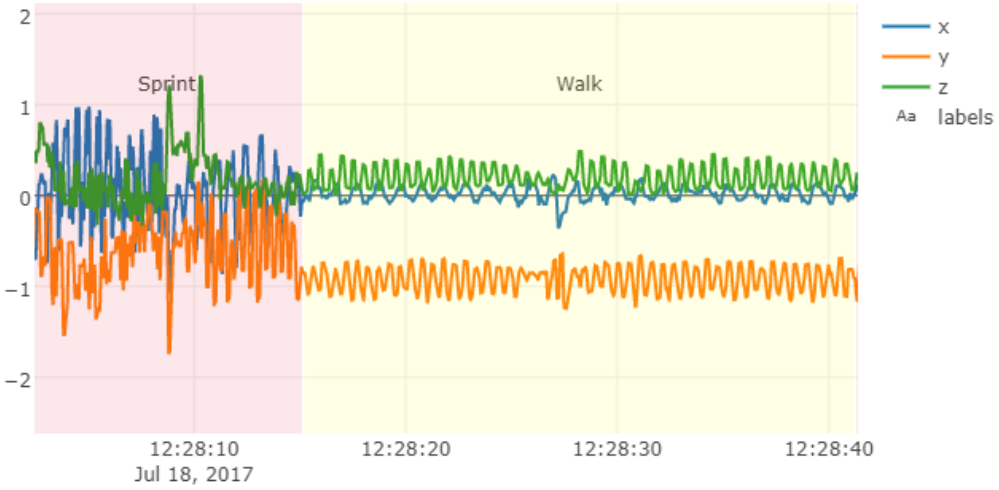


Figure 20. Accelerometer x, y and z signals produced for the Sprint and Walk classes

The jog and cruise motions struggled in comparison with a significant portion of jog being classified as cruise and vice versa. This was perhaps because there is not a great deal of difference between the two movements. Figure 21 shows the accelerometer x, y and z signals produced for each, this highlights that visually there was not a great difference between the two motions signals. This would make the job of the classifier to correctly predict a class much more difficult.

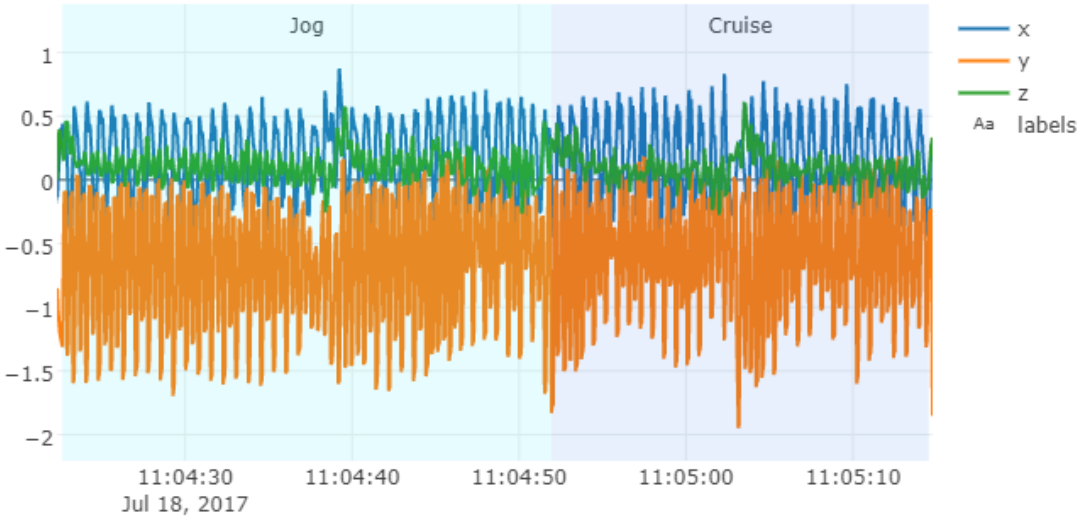


Figure 21. Accelerometer x, y and z signals produced for the jog class and cruise class

One solution to this specific class problem would be to combine these two classes into one. It would indefinitely improve the performance of the classifier. However, for use in professional sport analytics it will be more beneficial to have these as different movements. Participants commented on the difficulty to switch between these two movements in the routine, breaking up the routine for collecting data more by adding sprints and walks in between the jog and cruise motions could help the classifier deal with transitions between the movements better.

The cruise movement also had a significant proportion that would commonly be misclassified as a sprint. This is likely to be due to the definition of a cruise being at 75% of the pace of a sprint (see Appendix A for definition). Comparing it to the signal for the sprint in Figure 20 shows there are parts of the signal for the sprint which are very similar to the cruise. The data consists of movements from different participants and this is likely to have had an impact here. The participant who produced the slowest sprint and the participant with the fastest cruise could have produced very similar signals during the data collection routine. The LSTM RNN would struggle to learn the difference between them properly if this was the case.

The vertical jump, horizontal jump and stand data also did not classify as well as the walk and sprint classes. Figure 22 shows the accelerometer signals for these three classes.

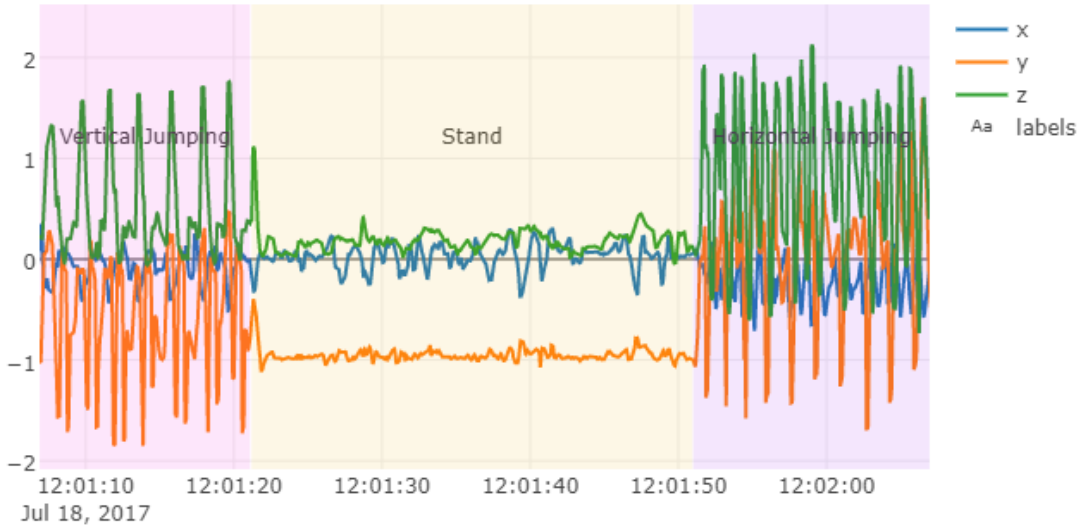


Figure 22. Accelerometer x, y and z signals produced for the Vertical Jump, Stand and Horizontal Jump classes

While it is very obvious (visually) the stand is different from both jump classes, this signal shows the length of time it was performed for. When a participant was performing the vertical jumps, there would be a short period of standing between each of the jumps. Within a sliding window this data could easily reflect the data for the standing class. The same reasoning is reproduced for the horizontal jumps; there would definitely be a moment where a participant was standing. This could partly contribute to the LSTM misclassifying some of the jumps as stands.

Across many of the different tables presented above the accuracy rate and the recall commonly have the same values. The recall is the sensitivity of the classifier, as discussed in section 2.5.3; it is the classifier's ability to detect all of the positive samples in the data. High accuracy rates have been achieved in this exploratory research so a high recall denoting few false negatives made by the classifier is not too surprising.

The reasoning behind altering the window size was to observe the performance of the vertical jump, horizontal jump and stand. However, the performance of the algorithm decreased and additionally the classification of the highlighted movements was much poorer. The next stage of this would be to increase the window size and examine the effect it would have on all the movement classes. Figure 23 shows the confusion matrix normalised to the total percentage of the test data to allow a more direct comparison between this and Figure 19. The jog and cruise classes are relatively similar to the data of which had a window size of 256 as is the

walk class. The sprint has performed slightly poorer. The stand looks to have improved however the jump classes are still easily misclassified.

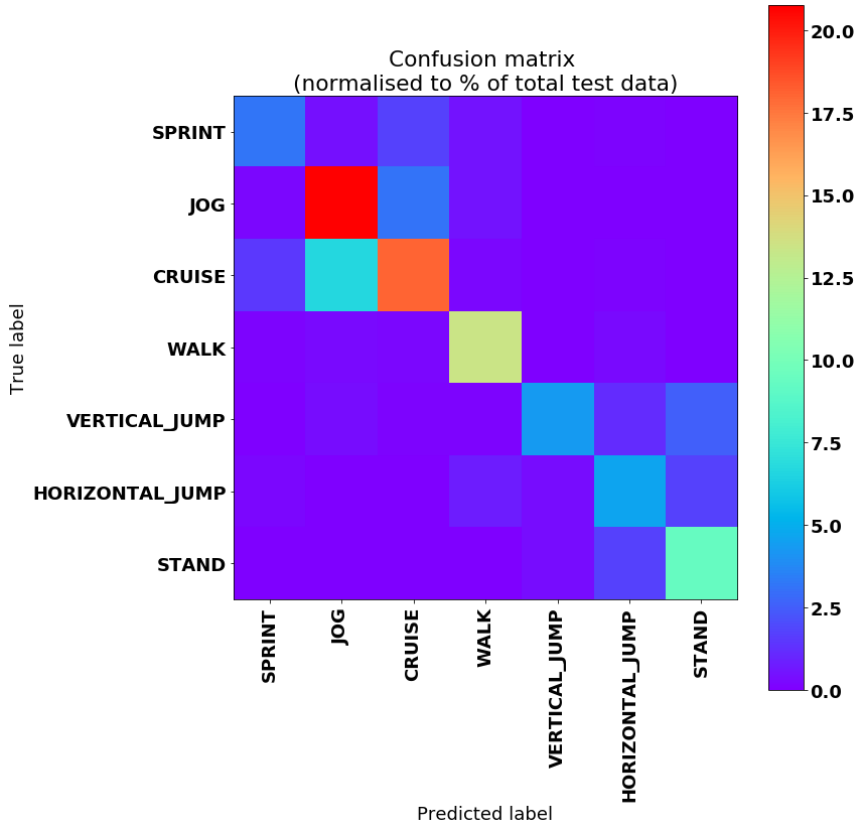


Figure 23. The confusion matrix for the optimal result obtained during training with the window size = 128

Another combination that also achieved a very high performance was the accelerometer only signals with 31 hidden layers. The hypothesis was that a poorer result would be obtained due to the data provided by the gyroscope being absent. However, this was not the case and an accuracy measure of approximately 80% was obtained. This was very surprising as the gyroscope supplies an insight into the rotations of the movements and it was presumed without this a great deal of information would be lost. The paper presented by A.J. Casson et al. researched movement analysis with accelerometers and gyroscopes for heart rate monitoring purposes. It was suggested that "it may be possible to dynamically switch between using accelerometer and gyroscopic data at different points in time." A period of sampling using gyroscope data is desirable due to the higher power consumption of a gyroscope in comparison to an accelerometer. Using this data to augment the motion estimation could produce a much more powerful and precise classifier. [70]

7.3.1 Analysis of the sessions training progress

During the training of the neural network the accuracies and losses for the training and test data was kept track of. Figure 24 below shows the results of this on a graph. It represents the accuracies (the green lines) and losses (blue) of the optimal configuration.

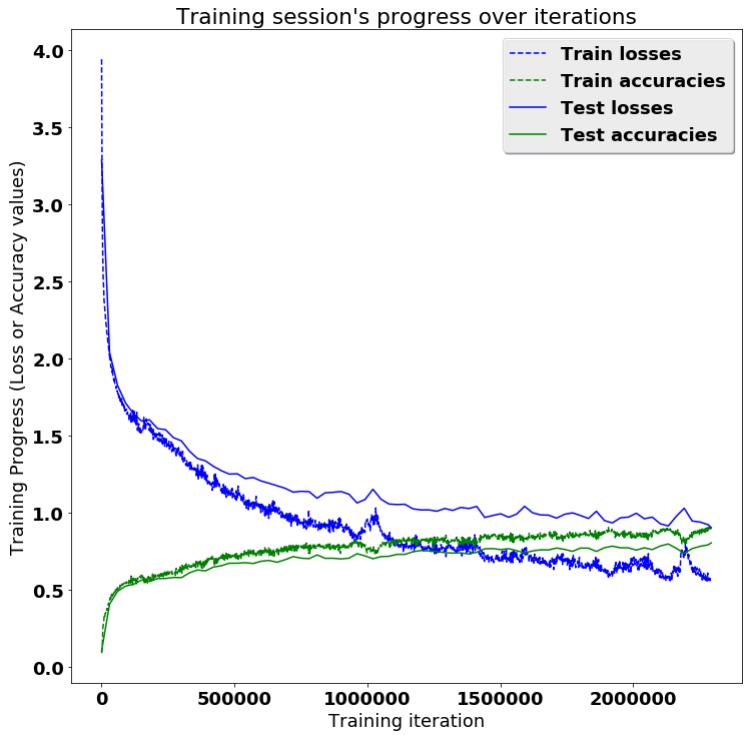


Figure 24. The progress of trainings session for the optimal configuration of the classifier

The blue dashed curve shows that the losses decays exponentially as the number of training iterations increase. It is also a relatively smooth curve for the training data with very minor spikes. The test data also decreases, but begins to level off resulting in a higher loss rate than the training data (which is always to be expected).

The accuracies increase as the number of iterations increase, as depicted by the green curves. The line for the training losses falls below the line for the accuracies at roughly around 1250000 iterations. An ideal situation would show both the test and training losses below the accuracy curves as time went on. The space between the training losses and test losses curve would also be much closer than the resulting graph in a perfect scenario.

Figure 25 shows the same graph for the poorest configuration unearthed during training; it was the no gravity signals feature set with 31 hidden layers (see Table 9). The curve does not have as rapid a decay as that in Figure 24. It is also much less smooth, with some nasty random spikes. Occasionally these random spikes would mean that a fault had occurred during training, such as a hardware fault or the AMI resources would be exhausted. However, it was

more likely the result of a poorer combination of hyperparameters and features causing the algorithm to not learn as well. The distance between the training and test losses also shows that this exact model has failed to learn to generalise to the test data.

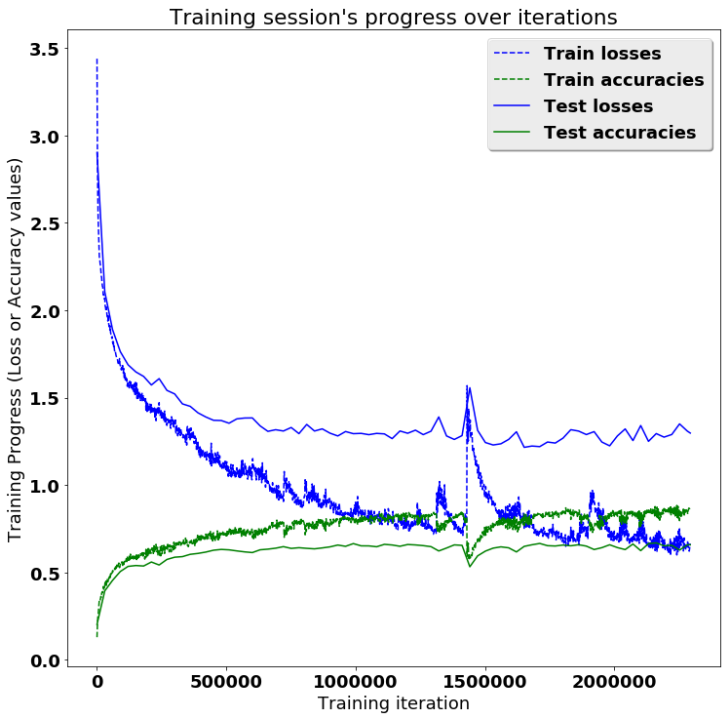


Figure 25. The progress of trainings session for the poorest configuration of the classifier that was discovered

8 Conclusion

8.1 Summary

The work carried out in this project was (as a whole) a huge success as all of the objectives were met in a relatively short time frame given the scale of the work. Initially, a considerable portion of time was used to research new methodologies and technologies, such as the Tensorflow, Scikit-learn and Pandas libraries for Python.

This research was conducted with a view to finding a solution to the motion sensor classification problem, and fully utilising the capabilities of the data available to GRN. This venture encompassed numerous objectives and challenges: designing an implementation for a machine learning algorithm capable of classifying movements, deploying techniques for visualising the end result and signals during the process, and improving the performance of the classifier.

The scope and objectives of this work outlined in Section 1.2 were met by firstly conducting research in the Human Activity Recognition (HAR) field. The findings of various learning algorithms and programming approaches were used to create the development cycle for the classifier at GRN. This included implementing signal processing techniques, segmenting the data, feature engineering, and the machine learning algorithm itself.

The final output was a prototype of a solution with capabilities in classifying select movements. The final and highest accuracy achieved for this classifier was 80.4%; meaning that of the seven movements classified the Long Short-Term Memory Recurrent Neural Network was able to correctly classify 80.4% of the data instances. This technical paper is also part of the final output. The steps and decisions made are all well documented to enable GRN to understand, replicate, and improve upon this work.

Finally, many problems can be solved by using Data Science applications. Each problem tends to have an aspect that is unique to that problem alone, and it is highly likely that it has not been seen before. As such, it is not possible to know which techniques, methods or algorithms to deploy initially or even whether the problem can be solved effectively using the available data. On a personal level, I have found this journey to be a very rewarding and exciting one. The steep learning curve was challenging, yet a real sense of pride and achievement was gained by obtaining a powerful prototype in such a short time-frame. I look forward to facing similar problems as a Data Scientist in the future.

8.2 Critical Evaluation

Although this project as an exploratory research activity was on the whole very successful, the final product produced still has a lot of room for improvement. The following is a brief list of proposals for future work that will cover limitations and ways to improve in addition to more work that would have been covered if time permitted.

1. As this research was an exploratory investigation into the quality of data available at GRN and what it was capable of doing, the motions selected were basic movements. It was important to start with this and produce an algorithm capable of correctly classifying these motions. However, the data collection activity discussed in section 3.1 reflects that throwing and catching (passing ring) movements were also captured. Due to time constraints, this data was not labelled and therefore not trained by the classifier. The next stage would be to label this data and feed it into the LSTM RNN.

2. Currently the company's products are marketed towards Rugby Union players, after expanding upon the basic motions covered in this research the next progressive stage would be to include more complex motions. Line outs, mauls, scrums, tackles and rucks are some of the motions that should be investigated making the product more attractive to potential customers.

GRN also intends to begin promoting their software to other sports. The basic motions captured in this research activity provide a solid groundwork for branching into more team sports and investigating more specific motions.

3. In order to make the classifier more robust it is good practice to add to the data used in training to make it larger. Due to how time-consuming the process of labelling the data was unfortunately this was not carried out. Data from different sources (i.e. different participants) will also add to the robustness of the classifier. Including data from different participants will change the demographic of the sample; it is quite probable that the demographic of the sample will have a significant effect on the classifier performance. The technique and performance level will differ producing a slightly different signal.

4. An extension of this work would be to down sample the size of the data. Changing the sampling rate from 100Hz to 50Hz would reduce the costs of storage in a database however, could have a significant impact on the performance of the algorithm. Too much information could be lost by halving the data and the LSTM could struggle to correctly classify. It would be another interesting research activity to investigate the impact down sampling would have on the data and the performance of the classifier.

5. In adding more data to this work one problem that could arise is the "Bias Variance Trade-off"; which means that the algorithm could over fit or under fit to the data and fail to general-

ise; see a full explanation of this in [71]. One way to avoid this problem would be to introduce dropout or other similar regularisation methods to avoid over-fitting the data.

5. The LSTM RNN proved very successful in being able to handle the data; however it would still be beneficial to try different neural networks, machine learning algorithms or different configurations of LSTMs. For example, a Bidirectional LSTM or a deep CNN with LSTM units built on top of it.

6. In theory, using two trackers could have been a good idea and proved useful in measuring the effectiveness of the IMUs. However, it created extra work with applying the signal processing techniques to double the number of files and then merging. Personal opinion dictates that no advantage was gained from doing this. The bottom device usually produced a much more accurate signal due to the fact it was held tightly in the correct position as discussed in section 4.2. Recreating this work with only the bottom devices could be a beneficial activity.

7. It was also discovered very late on in the process (from a different, much smaller attempt at data collection) that tracker number 2 was faulty and not logging the data correctly. Upon further investigation, the signal for this tracker was not what was expected and differed greatly compared to the signal produced from other trackers. This is highly likely to have had a significant impact on the results. Removing this tracker's data will improve the quality of the dataset and thus the resulting classification and respective accuracies.

8. Unfortunately, there was a delay in receiving the labelled dataset of training the model. As a Data Scientist a logical solution to overcome this blocking issue (due to time constraints) was to use a publicly available data set. A dataset on classifying different motions made available by the University of California [72] was a good alternative. The LSTM algorithm could be tested using this and the format of this data fit well with the network. The existing data already available at GRN was used to analyse the signal processing techniques before the dataset became available.

9. There was however not enough time to fully explore different feature engineering aspects suitable for use in this work. The results of the FFT were not as good as hoped. A Periodogram or Spectrogram, Discrete Wavelets Transforms and Dynamic Time Warping are among a few methods for time-series segmentation and feature engineering that could improve the resulting accuracy measures of the LSTM.

10. Another intention is to eventually use the trackers for group performance rather than just examining all individuals' performance. In regards to classifying some more complex motions group activities such as a line up could be investigated with the classifier. Providing insight into a team's performance in addition to an individual's performance would make any of the performance management aspects of GRN's services very attractive to customers.

8.3 Recommendation of Database

When large quantities of data are available, a database makes retrieving and storing the data a much easier task. A database will structurally organise and capture large quantities of data by inputting, storing, retrieving and managing information. It saves a data scientist (or any programmer using data) time, which in turn would save the company money. Working with such large CSV files was very challenging at times and luckily the Pandas library was able to cope well with reading in a large number of files to Python.

Postgres is an object relational database management system that would be a good solution. It uses SQL query language which means that complex joins may be involved when accessing the data; however, it is a mature and stable piece of software which means it will be very dependable. [75]

Relational databases are very structured and would be able to cope well with the structured data at GRN. However, it is known that it is not as effective for updating compared to non-relational database solutions.

Depending upon the size of the data that is being captured Cassandra could be a better alternative. Apache Cassandra is a free and open source, column family, NoSQL database. It has a simple architecture and robust support is available for clusters by scanning multiple datacentres. It also has no single point of failure so the data is in no danger of being lost. It has been likened to SQL solutions due to the very similar language CQL used to access the data but does not have the complete functionality of the SQL language, such as group by commands etc. Cassandra will be efficient for handling very large volumes of data; however, was not purpose build for time-series data. [76]

InfluxDB is a database purpose built for time-series data. Once the data has been timestamped it can be easily stored in the InfluxDB database. It has the ability to handle millions of data points per second and also has the ability to automate a process for down sampling data. [77] Other database management solutions that are purpose built for time-series data include Druid, Prometheus, Open TSB, Riak-TS, and Elasticsearch. [76]

References

[1] L. Steinberg, "CHANGING THE GAME: The Rise of Sports Analytics", Forbes.com, 2017. [Online]. Available: https://www.forbes.com/sites/leighsteinberg/2015/08/18/changing-the-game-the-rise-of-sports-analytics/#7dd2d1954c1f.

[2] "Moneyball", wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Moneyball.

[3] Z. Technologies and Z. Technologies, "Sponsored: The sports industry is following pro football players' every move", Quartz, 2017. [Online]. Available: https://qz.com/941307/for-the-nfl-and-industry-real-time-data-is-a-game-changer/

[4] "Technology Will Change the Future of Professional Sports!", The Medical Futurist, 2017. [Online]. Available: http://medicalfuturist.com/technology-changes-the-future-of-professional-sports/.

[5] A. Cave, "The potential of sport: a £20 billion industry", The Telegraph, 2017. [Online]. Available: http://www.telegraph.co.uk/investing/business-of-sport/potential-of-sport-20billion-industry/.

[6] B. Marr, "15 Mind Boggling Facts About Wearables In 2016", Forbes.com, 2017. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2016/03/18/15-mind-boggling-facts-about-wearables-in-2016/#91688e127323.

[7] "Global Rugby Network | About", Globalrugbynetwork.com, 2017. [Online]. Available: https://globalrugbynetwork.com/about.

[8] "Data Economy Demands New Approach Antitrust Rules World's Most Valuable Resource", economist.com, 2017. [Online]. Available: http://www.economist.com/new/leaders/21721656-data-economy-demands-new-approach-antitrust-rules-worlds-most-valuable-resource.

[9] S. Pudwell, "Machine Learning Is The Future Of Sports Data", Silicon UK, 2017. [Online]. Available: http://www.silicon.co.uk/data-storage/bigdata/machine-learning-data-206762?inf_by=59c22d4f671db8e5108b47e1

[10] S. Patel, H. Park, P. Bonato, L. Chan and M. Rodgers, "A review of wearable sensors and systems with application in rehabilitation", Journal of NeuroEngineering and Rehabilitation, vol. 9, no. 1, p. 21, 2012.

[11] A. Powell, "Catapult Sports GPS Tracking Comes to Soccer", Gear Patrol, 2017. [Online]. Available: https://gearpatrol.com/2016/08/31/catapult-gps-system-soccer/.

[12] G. Mairs, "Revealed - rugby's secret science lab", Telegraph.co.uk, 2017. [Online]. Available:

http://www.telegraph.co.uk/sport/rugbyunion/international/england/8527431/Revealed-rugbys-secret-science-lab.html.

[13] “Wearable Technology for Rugby - Muddy Rhino", Muddyrhino.com, 2017. [Online]. Available: http://muddyrhino.com/wearable-technology-rugby/.

[14] "Machine Learning: What it is and why it matters", Sas.com, 2017. [Online]. Available: https://www.sas.com/en_gb/insights/analytics/machine-learning.html.

[15] "Why Is Machine Learning So Popular", Quora.com, 2017. [Online]. Available: https://www.quora.com/Why-is-machine-learning-so-popular.

[16] J. Brownlee, "Supervised and Unsupervised Machine Learning Algorithms", Machine Learning Mastery, 2017. [Online]. Available: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/.

[17] B. Marr, "Supervised v Unsupervised Machine Learning: Whats the Difference", Forbes.com, 2017. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2017/03/16/supervised-v-unsupervised-machine-learning-whats-the-difference/#50032fcb485d.

[18] "Introduction to Semi-Supervised Learning", MIT Press.

[19] "Activity Recognition Challenge - Dataset | Opportunity", Opportunity-project.eu, 2017. [Online]. Available: http://www.opportunity-project.eu/challengeDataset.

[20] A. Akin, S. Bosch, M. Marin-Perianu and P. Havinga, "Activity recognition using inertial sensing for healthcare, wellbeing and sports applications: A survey", Architecture of computing systems (ARCS), 2010, no. 23, pp. 1-10, 2010.

[21] F. Korbinian, V. Nadales, M. Josefa, P. Robertson and M. Angermann, "Reliable real-time recognition of motion related human activities using MEMS inertial sensors", 2010.

[22] C. Saisakul, A. Atkins and H. Yu, "Activity classification using a single wrist-worn accelerometer", Software, Knowledge Information, Industrial Management and Applications (SKIMA), no 5, pp. 1-6, 2011.

[23] R.A. Samir, D.A. Rafeldt and T.L. Uhl, "Wearable IMU for Shoulder Injury Prevention in Overhead Sports", Sensors 16, no. 11, p. 1847, 2016.

[24] R. Palakodety, "Activity recognition using accelerometer data", Computer Science Conference for University of Bonn Students, 2015.

[25] C.Y. Yong, R. Sudirman, N.H. Mahmood and K.M. Chew, "Motion Classification Using Proposed Principle Component Analysis Hybrid K-Means Clustering", Engineering 5, no. 5, p. 25, 2013.

[26] D.T. Morris, S. Saponas, A. Guillory and I. Kelner, "RecoFit: using a wearable sensor to find, recognize, and count repetitive exercises", Proceedings of the 32nd annual ACM conference on Human factors in computing systems, pp. 3225-3234, 2014.

[27] P. Matthias, "Classification of Human Whole-Body Motion using Hidden Markov Models", arX, vol iv, pp. 1605.01569, 2016.

[28] R. DeVaul and S. Dunn, "Real-time motion classification for wearable computing applications", 2001.

[29] L. Bao and S. Intille, "Activity recognition from user-annotated acceleration data", Persuasive Computing (2004), pp. 1-17, 2004.

[30] G. Bailador, D. Roggen, G. Tröster and G. Triviño, "Real time gesture recognition using continuous time recurrent neural networks", ICST 2nd international conference on Body area networks, p. 15, 2007.

[31] T.T. Um, V. Babakeshizadeh and D. Kulic, "Exercise Motion Classification from Large-Scale Wearable Sensor Data Using Convolutional Neural Networks", 2016.

[32] X. Long, B. Yin and R. Aarts, "Single-accelerometer-based daily physical activity classification", In Engineering in Medicine and Biology Society. EMBC 2009. Annual International Conference of the IEEE, pp. 6107-6110, 2009.

[33] F.J. Ordóñez and D. Roggen, "Deep convolutional and LSTMrecurrent neural networks for multimodal wearable activity recognition", Sensors 16, no. 1, p. 115, 2016.

[34] C. Stergiou and D. Siganos, "Neural Networks", Doc.ic.ac.uk, 2017. [Online]. Available: https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html.

[35] R. Jazefowicz, W. Zaremba and H. Sutskever, "An Empirical Exploration of Recurrent Network Architectures", mlr press, vol. 37, 2015.

[36] A. Karpathy, "The Unreasonable Effectiveness of Recurrent Neural Networks", github.io, 2015. [Online]. Available: http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

[37] J. McCaffrey, "Understanding Neural Network Batch Training: A Tutorial", Visual Studio Magazine, 2014. [Online]. Available: https://visualstudiomagazine.com/articles/2014/08/01/batch-training.aspx.

[38] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory", Neural Computation, vol. 9, no. 8, pp. 1735-1780, 1997.

[39] M. Sundermyer, R. Schlüter and H. Ney, "LSTM Neural Networks for Language Modeling", Thirteenth Annual Conference of the International Speech Communication Association, 2012.

[40] A. Colah, "Understanding LSTM Networks", Colah.github.io. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[41] "Pandas 0.20.3: Python Package Index", Pypi.python.org, 2017. [Online]. Available: https://pypi.python.org/pypi/pandas/.

[42] "Signal processing (scipy.signal) — SciPy v0.19.1 Reference Guide", Docs.scipy.org, 2017. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/signal.html.

[43] "MATLAB - MathWorks", Uk.mathworks.com, 2017. [Online]. Available: https://uk.mathworks.com/products/matlab.html.

[44] "What are the Comparative Pros and Cons of Using Python MATLAB Octave and R for Data Analysis and Machine Learning", Quora.com, 2017. [Online]. Available: https://www.quora.com/What-are-the-comparative-pros-and-cons-of-using-Python-MATLAB-Octave-and-R-for-data-analysis-and-machine-learning.

[45] "R: What is R?", R-project.org, 2017. [Online]. Available: https://www.r-project.org/about.html.

[46] "R: Geosphere package", R-project.org, 2017. [Online]. Available: https://cran.r-project.org/web/packages/geosphere/geosphere.pdf.

[47] "Python Release Python 3.5.2", Python.org, 2017. [Online]. Available: https://www.python.org/downloads/release/python-352/.

[48] "conda 4.0.2 : Python Package Index", Pypi.python.org, 2017. [Online]. Available: https://pypi.python.org/pypi/conda/4.0.2.

[49] "Project Jupyter", Jupyter.org, 2017. [Online]. Available: http://jupyter.org/.

[50] A. Anisin, "Comparing Python and R for Data Science", Data Science Blog by Domino, 2017. [Online]. Available: https://blog.dominodatalab.com/comparing-python-and-r-for-data-science/.

[51] "TensorFlow", TensorFlow, 2017. [Online]. Available: https://www.tensorflow.org/.

[52] "Amazon Web Services (AWS) - Cloud Computing Services", Amazon Web Services, Inc., 2017. [Online]. Available: https://aws.amazon.com/.

[53] "AWS Marketplace: Deep Learning AMI Amazon Linux Version", Aws.amazon.com, 2017. [Online]. Available: https://aws.amazon.com/marketplace/pp/B01M0AXXQB.

[54] S. Wang and X. Yao, "Multi-Class Imbalance Problems: Analysis and Potential Solutions", IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS, no. 424, pp. 1119-1130, 2012.

[55] "MPU-9250 Product Specification Revision 1.1", invensense.com, 2015. [Online]. Available: https://www.invensense.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf.

[56] "WHAT IS THAT BUMP AT THE BACK OF THE LIONS JERSEYS? | Catapult UK", Catapultsports.com, 2017. [Online]. Available: http://www.catapultsports.com/uk/media/what-is-that-bump-at-the-back-of-the-lions-jerseys/.

[57] D. Anguita, A. Ghio, L. Oneto, X. Parra and J. Reyes-Ortiz, "A Public Domain Dataset for Human Activity Recognition using Smartphones", ESANN, 2013.

[58] "scipy.signal.medfilt — SciPy v0.19.1 Reference Guide", Docs.scipy.org, 2017. [Online]. Available:

https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.medfilt.html.

[59] J. Parkka, M. Ermes, P. Korpipaa, J. Mantyjarvi, J. Peltola and I. Korhonen, "Activity classification using realistic data from wearable sensors", IEEE Transactions on information technology in biomedicine, vol. 10, no. 1, pp. 119-128, 2006.

[60] "scipy.signal.butter — SciPy v0.14.0 Reference Guide", Docs.scipy.org, 2017. [Online]. Available: https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.signal.butter.html.

[61] "scipy.signal.filtfilt — SciPy v0.14.0 Reference Guide", Docs.scipy.org, 2017. [Online]. Available: https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.signal.filtfilt.html.

[62] J.M. Molina, J. Garcia, A.C. Bicharra Garcia, R. Melo and L. Correia, "Segmentation and classification of time-series: Real case studies", International Conference on Intelligent Data Engineering and Automated Learning. Springer, 2009.

[63] "What is the difference between numpy.fft and scipy.fftpack?", Stackoverflow.com, 2017. [Online]. Available: https://stackoverflow.com/questions/6363154/what-is-the-difference-between-numpy-fft-and-scipy-fftpack.

[64] "Linear algebra (numpy.linalg) — NumPy v1.13 Manual", Docs.scipy.org, 2017. [Online]. Available: https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.linalg.html.

[65] D. Kingma and J. Ba, "Adam: A method for stochastic optimization", arXiv, no. 14126980, 2014.

[66] J. Brownlee, "Why One-Hot Encode Data in Machine Learning?", Machine Learning Mastery, 2017. [Online]. Available: https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/.

[67] A. Ng, "Sparse Autoencoder", web.stanford.edu, 2017. [Online]. Available: http://web.stanford.edu/class/cs294a/sparseAutoencoder_2011new.pdf.

[68] C. Raffel, "Neural Network Hyperparameters", Colinraffel.com, 2017. [Online]. Available: http://colinraffel.com/wiki/neural_network_hyperparameters.

[69] "sklearn.model_selection.train_test_split — scikit-learn 0.19.0 documentation", Scikit-learn.org, 2017. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

[70] A. Casson, A. Vazquez Galvez and D. Jarchi, "Gyroscope vs. accelerometer measurements of motion from wrist PPG during physical exercise", ICT Express, vol. 2, no. 4, pp. 175-179, 2016.

[71] S. Fortmannn-Roe, "Understanding the Bias-Variance Tradeoff", Scott.fortmann-roe.com, 2017. [Online]. Available: http://scott.fortmann-roe.com/docs/BiasVariance.html.

[72] "UCI Machine Learning Repository: Human Activity Recognition Using Smartphones Data Set", Archive.ics.uci.edu. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones

[73] guillaume-chevalier, "LSTM-Human-Activity-Recognition", GitHub, 2016. [Online]. Available: https://github.com/guillaume-chevalier/LSTM-Human-Activity-Recognition.

[74] M. Deutsch, G. Kearney and N. Rehrer, "Time – motion analysis of professional rugby union players during match-play", Journal of Sports Sciences, vol. 25, no. 4, pp. 461-472, 2007.

[75] "Ask HN: What DB to use for huge time series?", Hacker News, 2017. [Online]. Available: https://news.ycombinator.com/item?id=8368509.

[76] Netsil, "A Comparison of Time Series Databases and Netsil's Use of Dru-id", blog.netsil.com, 2017. [Online]. Available: https://blog.netsil.com/a-comparison-of-time-series-databases-and-netsils-use-of-druid-db805d471206.

[77] "InfluxData: The leading Platform for Monitoring & Analytics", InfluxData, 2017. [Online]. Available: https://www.influxdata.com/products/.

Appendix A

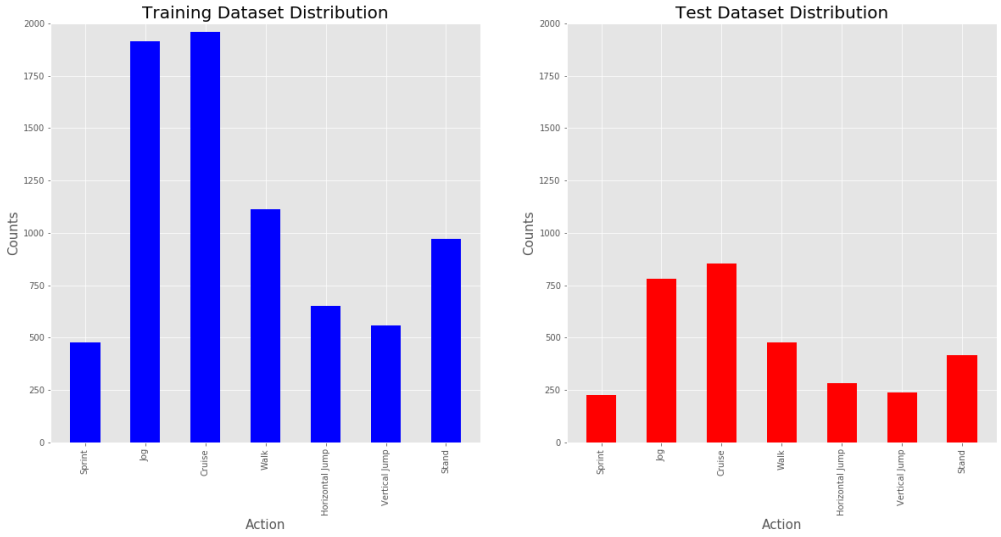
The motions being classified in this work are further explained by the following operational definitions [74]:

- *Stand* - standing without travelling any further forward or backward, the participant should not be involved in any other motions. Small natural movements that were not purposeful were allowed, such as shaking out legs, stretching and turning sideways.
- *Walk* – walking forwards at a pace that felt natural for the participant without much haste. One foot was in contract with the ground at all times.
- *Jog* – running in a forwards direction at a slower pace (no particular haste) and no arm drive
- *Cruise* – running in a forwards direction, with a pace faster than a jog but still not exerting to maximal effort (3/4 pace).
- *Sprint* – high impact running, in a forwards direction, participant should do so with maximal effort.
- *Vertical jump* – jumping in an upwards direction without travelling forwards or backwards.
- *Horizontal jump* – jumping whilst moving in a forwards direction with arm drive.

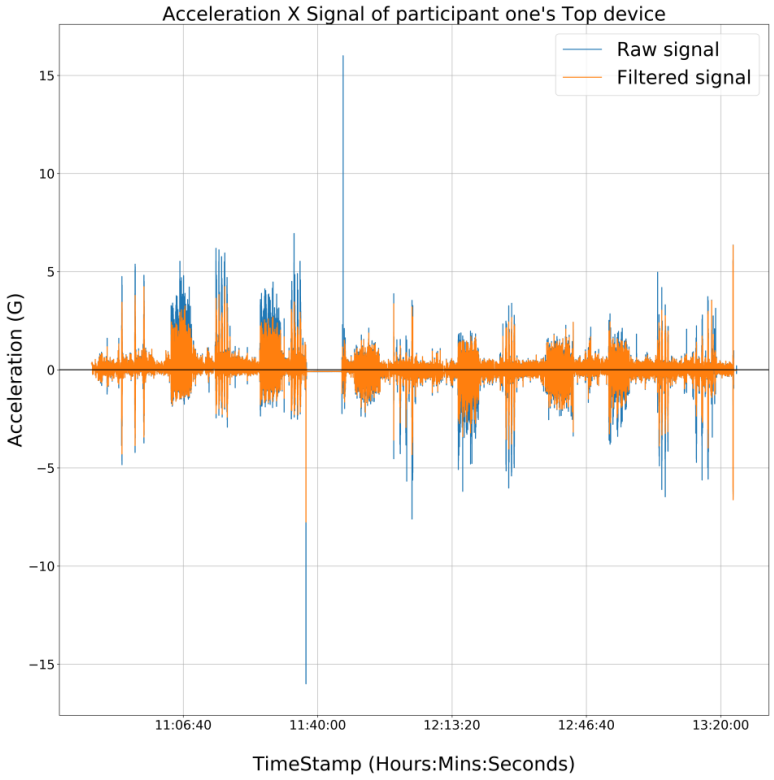
Appendix B

Distribution of the Training and Test Data

Bar Plots of the distribution of classes in the Training and Test Dataset



Raw accelerometer X signal vs the final filtered signal of participant one's top device



Appendix C

Python code for the Butterworth Filters

```
def butter_lowpass(self, cutoff, nyq_freq, order=3):

    """ Building the Butterworth filter with the frequency.
        The cutoff freq is the number of Hz that is used to filter out any
        data above this threshold
        Nyquist frequency (nyq_freq) is 1/2 of the sampling rate
    """

    normal_cutoff = float(cutoff) / nyq_freq
    b, a = signal.butter(order, normal_cutoff, btype='lowpass')

    return b, a


def butter_lowpass_filter(self, data, cutoff_freq, nyq_freq, order=3):

    """ This function will apply the 3rd order low pass Butterworth filter
        to the specified data.
    """

    b, a = self.butter_lowpass(cutoff_freq, nyq_freq, order=order)
    y = signal.filtfilt(b, a, data)

    return y
```

Python code for the LSTM algorithm

```
def LSTM_RNN(_X, _weights, _biases):

    """ The LSTM Network.
        Function returns a TensorFlow LSTM (RNN) artificial neural
        network from given parameters. Moreover, two LSTM cells are
        stacked which adds deepness to the neural network.
    """

    # input shape: (batch_size, n_steps, n_input)
    # permute (swap) n_steps and batch_size
    _X = tf.transpose(_X, [1, 0, 2])

    # Reshape to prepare input to hidden activation
    # new shape: (n_steps*batch_size, n_input)
    _X = tf.reshape(_X, [-1, n_input])

    # Linear activation
    # new shape: n_steps * (batch_size, n_hidden)
    _X = tf.nn.relu(tf.matmul(_X, _weights['hidden']) + _biases['hidden'])
    _X = tf.split(_X, n_steps, 0)

    # Define two stacked LSTM cells (two recurrent layers deep)
    lstm_cell_1 = tf.contrib.rnn.BasicLSTMCell(n_hidden,
                        forget_bias=1.0, state_is_tuple=True)
    lstm_cell_2 = tf.contrib.rnn.BasicLSTMCell(n_hidden,
                        forget_bias=1.0, state_is_tuple=True)

    lstm_cells = tf.contrib.rnn.MultiRNNCell(
                             [lstm_cell_1, lstm_cell_2], state_is_tuple=True)

    # Get LSTM cell output
    outputs, states = tf.contrib.rnn.static_rnn(lstm_cells, _X,
                                                dtype=tf.float32)

    lstm_last_output = outputs[-1]

    # Linear activation
    return tf.matmul(lstm_last_output, _weights['out']) + _biases['out']
```