Utrecht University

Master's Thesis

Classifying posture and activity using a wearable device

Author:
Pim Sauter

Supervisor:
Dr. A.J. Feelders

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the field of

Algorithmic data analysis

September 6, 2017

Abstract

For a patient monitoring system to work, it is important to keep track of a patient's physiological signals, such as heart rate and blood pressure. Equally important is the availability and analysis of data that can add context to the physiological signals. In this thesis, accelerometer and gyroscope data collected from a medical wearable device are used to infer the posture and activity of the wearer.

A simple, threshold based method is presented, that distinguishes between activity and rest and can distinguish between some postures. Additionally, a method is presented that can detect postural transitions, and these transitions are classified using different neural network architectures and other machine learning methods. A fully connected neural network with regularization achieves the best classification accuracy.

The same methods are applied to classify postures, activities and postural transitions. The best overall classification accuracy is achieved with a window-based random forest classifier, combined with an LSTM network over those windows. The thesis is concluded with a discussion about the usability of the methods in a real life application.

Contents

List of Abbreviations

HMM	**H**idden **M**arkov **M**odel
CRF	**C**onditional **R**andom **F**ield
ANN	**A**rtificial **N**eural **N**etwork
SGD	**S**tochastic **G**radient **D**escent
RNN	**R**ecurrent **N**eural **N**etwork
LSTM	**L**ong **S**hort **T**erm **M**emory

Chapter 1

Introduction

"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it"

With this quote, Mark Weiser started his 1991 landmark paper [41], where he coined the term *ubiquitous computing*, the concept of computing being all around us in everyday life. A quarter of a century has passed, and Weiser's vision is becoming more and more a part of our reality. Microprocessors are being integrated into everything, and our everyday life is no longer possible without this processing power all around. An important component in ubiquitous computing is *context awareness*. Dey and Abowd [10] describe it as *the ability to use context to provide relevant information and/or services to the user, where relevancy depends on the user's task.*

Microprocessors and sensors have become cheaper and smaller over the years, leading to a growth in context aware ubiquitous computing. Google is able to provide you with relevant (and sponsored) content, based on your location, search history, and many more parameters. Over the last few years, context aware computing has had an impact on many fields, including consumer health care. A notable example is the fitness tracker, that uses sensors (accelerometer, sometimes GPS) to deduce what activity the user is doing, and estimates parameters such as steps taken and energy expenditure.

In this thesis, a medical application of ubiquitous computing is discussed, which uses a novel wearable device to recognize posture and activity of the user. Section 1.1 introduces the context and motivation, and the research questions will be stated in Section 1.2. The research methodology will be explained in Section 1.3. Finally, Section 1.4 will give an outline of the rest of the thesis.

1.1 Motivation

After staying in a hospital, patients return to their homes and normal daily life. There is a risk assessment involved in sending a patient home; sending a patient home too soon, when a full recovery has not yet been made, could result in a dangerous situation. On the other hand, keeping a patient too long has drawbacks as well: it is costly to keep someone in the hospital, and it takes longer for the patient to return to their normal lives.

Continuous monitoring at the patient's home would solve this problem, but is often infeasible. Firstly, monitoring equipment that can provide accurate information about the patient's vital signs is often expensive, and hospitals do not have enough of this

equipment to provide for every patient in need. The equipment is usually heavy and bulky, and impedes patients from living their normal lives when attached to it. An example is an ECG monitor, which involves attaching 12 wired electrodes to different parts of the body.

The wearable device that is used in this thesis attempts to provide a solution to continuous monitoring of patients. It is cheap, which means it can be produced in bulk and provided to a large number of patients. It is very small (the size is about that of a matchbox), and can worn around the arm with an armband or on the chest with a chest patch, providing little to no inconvenience for the user. The device is equipped with multiple sensors to infer physiological data, such as heart rate, blood pressure and skin temperature.

The physiological data is collected at the patient's home and not in a hospital bed, which is where the need for context data arises. An observed increase in heart rate can be a dangerous event when the patient is in bed, or just a physical response when the patient is climbing the stairs. For this reason, the device is also equipped with an accelerometer and gyroscope, which will be used for the recognition of posture and activity. This data can then be combined with the physiological data, to provide a more accurate image of the state of the patient.

1.2 Research questions

Key to the success of continuous monitoring is the quality of the context information. This thesis is concerned with the inference of context information from data acquired with the wearable device. The central **research question** is:

Is it possible to accurately recognize posture and activity of an individual using accelerometer and gyroscope data acquired from a wearable device?

While there are many methods to achieve this task, special attention will be given to neural networks, as they have been shown to consistently outperform other machine learning methods at a variety of machine learning tasks. The **sub-question** is hence:

Do neural networks provide a better recognition of posture and activity, compared to other machine learning methods?

1.3 Methodology

Many approaches have been used to attempt the classification of posture and activity, cf. Chapter 2. Across the different approaches, there is a significant variation in achieved classification accuracy. It is difficult to compare these one to one, as there is a large number of parameters which varies among different approaches. The number of postures or activities that is chosen is an important factor, as recognizing a lot of activities will make the choice harder for any classifier, especially when activities exhibit similar movements. The number of accelerometers and their placement also has an impact; using more accelerometers can improve the result, as data from two or more devices gives info about the positions of the devices relative to each other.

In this thesis, the activity recognition task will be performed using a wearable device, equipped with an accelerometer and gyroscope, worn at the arm. The classifier will

have to work with data from unknown subjects, which means person specific training data is not available. Because of the high bandwidth of the accelerometer and gyroscope signals, it is infeasible to transmit them to a computer, to perform the classification there. This means the classification has to be performed on the device, which requires an efficient algorithm. Training data will be collected beforehand, which means model training can happen offline.

To collect the training data, a protocol has been designed, which includes different postures and activities (a detailed explanation of the protocol is given in Chapter 4). Data is collected on 27 participants, which is a regular number in activity recognition research [23]. On this dataset, a comparative study of machine learning algorithms will be performed, where classification accuracy and computational complexity are evaluated.

1.4 Outline

The remainder of this thesis will start with a discussion of related work in Chapter 2. The methods used in the research will be explained in Chapter 3. Chapter 4 covers the experiments that have been performed and their results. Chapter 5 concludes the thesis, provides a discussion and discusses possible future work.

Chapter 2

Related work

This chapter gives a short overview of previous work on activity recognition, and will be structured in a chronological manner: Section 2.1 discusses how data is acquired and processed by the different researchers, Section 2.2 explains the process of feature extraction, Section 2.3 examines the classifier choices made by previous researchers and Section 2.4 compares other design choices made by different researchers. To conclude, a few example applications are given in Section 2.5.

2.1 Data acquisition and processing

The first step of the classification process is data acquisition, using one or more sensors. Most commonly, one or more accelerometers are used for this task. An accelerometer measures acceleration in one (uniaxial), two (biaxial) or three (triaxial) dimensions. After the choice of accelerometer, the number of accelerometers and their locations on the body are important. In [38] three uniaxial accelerometers are used, mounted on different parts of the body. Most later researchers have opted for a single triaxial [24, 8, 1, 3] or biaxial [13] accelerometer. Roggen et al. [30] uses 72 sensors, including accelerometers, microphones and RFID chips, to form a sensor network for activity recognition. The resulting dataset is the OPPORTUNITY dataset, which is freely available at the UCI machine learning repository.

Acceleration data is usually sampled at a frequency between 20 and 100 Hz. It is not possible to classify single frames, as they do not contain enough information. To overcome this problem, data can be processed in windows [22, 8]. For each window, features are calculated from that window's signal, which are then used to classify the activity in that window. To improve results, a sliding window with overlap can be used [20]. In [4], the influence of window size on classification accuracy is investigated; a shorter window size is found to be beneficial for the activity recognition task.

Before extracting features, noise needs to be removed from the signal, which can be achieved using a low-pass filter, which removes high frequencies that have do not originate from body acceleration [2]. The last preprocessing step before feature extraction is to separate the body acceleration and gravity acceleration components. This can be done using another low-pass filter, using the fact that body acceleration is in a higher frequency range than gravitational acceleration [1].

2.2 Feature extraction

For posture classification, different features have been used in the literature. Popular choices are mean and standard deviation of the signal over the time window. The mean of the gravitational acceleration in a window can be used to distinguish between postures [38], where the standard deviation can be used to distinguish between static and dynamic activities. The correlation between different accelerometer axes can be used to distinguish activities that move in one dimension (such as walking) from activities that move in more directions (such as climbing the stairs) [8]. Other features include signal skewness and kurtosis [2], and shifted delta coefficients (slope of the signal in different points) [1].

Features are extracted not only from the time domain, but also the frequency domain, after performing an FFT. The frequency domain feature energy [20] can distinguish between periodic (such as walking) and non periodic activities. Another frequency domain feature is entropy, which measures the complexity of the signal. This can distinguish between movements of different complexities, such as cycling and walking [20]. Park et al. [28] use spectral energy and FFT magnitudes as features.

2.3 Choice of classifier

Earlier work [38, 24] uses threshold based approaches to distinguish between activities (e.g. static versus dynamic or between different postures). These simple classifiers are then used to make a hierarchical classification of activities. The downside to this approach is that it requires careful tuning of the thresholds, and is not easily extendable to detect more activities or deal with different datasets. Later works deploy more sophisticated machine learning techniques to accomplish the task. Allen et al. [1] use a Gaussian Mixture Model [1] to detect 5 postures and 3 movements, and find that their approach performs well: their classification rate of 90 % on their data is a considerable improvement over the 70 % rate by a threshold based approach. The model is first trained on a large data set of multiple subjects. Person specific data is then used to create a separate model for each subject, using Bayesian adaptation.

Anguita et al. [3] propose a hardware-friendly approach to activity recognition, with accelerometers from a mobile phone, where they use Support Vector Machines (SVM). The dataset from this research has been made publicly available and is often used as benchmark [2]. They achieve 89 % precision and recall using 17 FFT and time domain features. The same researchers extend this approach in another paper [2], by extracting a total of 561 features from accelerometer and gyroscope data. With these features, they classify activities correctly 96 % of the time. In [42], different models are compared (among others k-Nearest Neighbor (kNN), decision tree, logistic regression), and kNN is found to have the best overall classification performance. Gyroscope data is found to improve classification accuracy. In [43], a neural network is used to first distinguish between static and dynamic activities, and recognize activities and postures. The classification has an accuracy of 85 % for 6 different activities.

The classification methods mentioned before do not take into account that the activity data is sequential in nature, but a recording of someone's activity is basically a sequence of postures, activities and transitions. There exist methods to exploit this

[1] An explanation of most of the classifiers in this section can be found in [6].

temporal aspect, which have also been applied in the field of activity recognition. The canonical probabilistic model for temporal data is the Markov Model, where future states exclusively depend on the current state. When the states are not directly observed, a Hidden Markov Model (HMM) [34] can be used, where observed variables model the hidden variables. Mannini and Sabatini [23] employ an HMM to distinguish 3 postures and 4 dynamic activities, in order to exploit the sequential nature of the data. They find that Markov modeling improves the classification accuracy, when compared to non-sequential classifiers. A alternative approach to sequence classification is the Conditional Random Field (CRF). Vinh et al. [39] apply CRFs to the activity classification problem, with a fast implementation that makes use of the interdependency and duration of the activities.

A method that has been gaining in popularity the last decade is that of Recurrent Neural Networks (RNNs). They combine the power of neural networks with the ability to exhibit temporal behavior, and are widely used for time series prediction and classification [18]. RNNs are currently used in some of the most exciting applications in machine learning, such as computer vision and speech recognition and prediction. Ordóñez and Roggen [27] apply a deep convolutional RNN architecture to the previously mentioned OPPORTUNITY dataset. They achieve an F_1 score of 0.93, where the highest non-RNN score is 0.91 and the highest score achieved without using neural networks is 0.87.

2.4 Other design choices

Most research uses offline training and classification, because of the high computational load associated with the techniques involved. Park et al. [28] train their classifier offline, and use it for online classification on a mobile device. The approach used by Anguita et al. [3] also focuses on reducing the computational complexity, such that classification can be done on the device. To validate their classifiers, some researchers use conventional techniques such as a train/test split [3]. Others use leave-one-participant-out cross validation, where training is done on all subjects but one, and validation is done on the remaining subject [20]. This is done to avoid that the accuracy is improved by person-specific training. Some researches choose to use person specific data to improve classification accuracy [1]. This approach is not feasible for applications, because usually there is no labeled data available for a new test subject.

Transitions occur between activity segments, which are usually quite noisy and difficult to classify. Usually, this problem is solved by just removing the transition moments [20], because the objective of the research is limited to classify the postures and activities, and the duration of transitions is usually short. In [29], the transition problem is explicitly addressed by creating a transition-aware human activity recognition system.

2.5 Applications

The applications of activity classification are manifold, and fall within the framework of context aware computing. Li et al. [21] developed a fall detection algorithm, based on accelerometer and gyroscope data; the algorithm tries to determine if a transition between activities is intentional or not. If there is an unintentional transition between two activities, there could be a fall. An algorithm like this could be implemented in

a continuous monitoring system for the elderly, who have an increased risk of falling. Park et al. [28] use accelerometer data from a smartphone to classify where the device is located (in the pocket, bag, hand or at the ear). Apps could use this information to adapt their behavior (e.g. notification type) to the current state of the device. In the same paper, accelerometer data is used to make a prediction of walking speed . In [19], the data from a single gyroscope is used to make a pedometer (step count) algorithm.

In [36], the data from an accelerometer is used to first classify if someone is walking or stationary; in case of walking, the accelerometer data is used to construct a navigation system based on dead reckoning. Such a navigation system could be used in places where the GPS signal is bad, for instance in cities with skyscrapers. García-García et al. [13] combine the accelerometer data with heart rate data to make an assessment of physical activity intensity, which is a good indicator of energy expenditure. This is relevant for people who want to keep track of their fitness or weight, or want to live a healthier lifestyle in general.

Chapter 3

Methods

Posture classification is a sequential *supervised* learning problem; supervised because there is labeled training data, sequential because the accelerometer data has a temporal aspect and can thus be treated as a sequence of data points. In this chapter, an overview of the methods used for the posture classification task will be given. Section 3.1 introduces classification, one of the main tasks in machine learning. Section 3.2 features an introduction to sequence labeling, with an overview of different methods. Section 3.3 introduces the concept of a neural network, which can be used for classification. Recurrent Neural Networks, which are suitable for sequence classification, are introduced in Section 3.5.

3.1 Classification

In a supervised learning problem, the goal is to infer a model from labeled training examples, which can later be used to classify new data. A famous example is the recognition of handwritten digits [9], where a number between 0 and 9 has to be assigned to each handwritten sample. Each training example is a pair (x, y), where x is a vector containing the pixel values of the image of the handwritten digit, and $y \in \{0, \ldots, 9\}$ denotes the label. The model takes as input a new example x and predicts a label y. A sample of handwritten digits, ordered by label, is shown in Figure 3.1. Not all digits of the same class are written exactly in the same way, because different people have different handwriting. Because of this, it is not possible to simply use a fixed template for each digit and use this for classification.

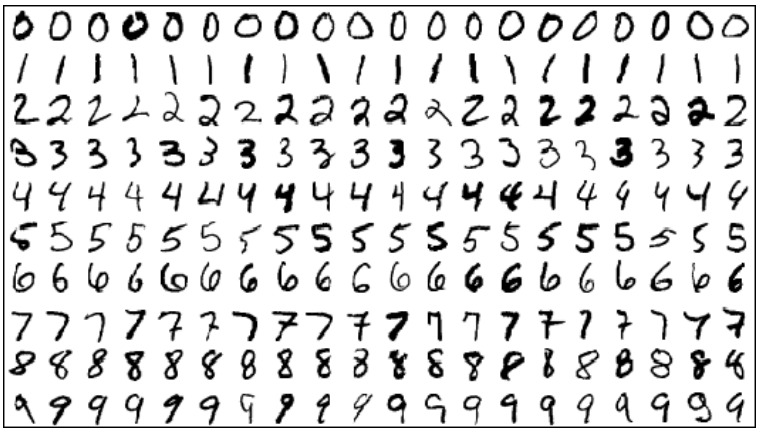

FIGURE 3.1: A sample of handwritten digits, ordered by label

To successfully perform a classification task, the model must detect patterns in the data. A problem that often arises is that the dimensionality of the problem if very high. For the handwritten digits, each training sample has $28 \cdot 28 = 784$ pixels, each with a grayscale value between 0 and 255. With 255^{784} possible images, it is safe to assume that none of the training examples are exactly the same. The key to finding patterns then is to reduce the dimensionality, by feature extraction. For the digits, a possible feature is 'amount of ink used', which can be easily computed by summing over all grayscale values in an image. This feature will not give a good classification rate on its own, but can distinguish between certain digits, such as 0 and 1. In many approaches to classification, the features are handcrafted from the data, which often requires a certain domain knowledge. These features are then fed to an algorithm that attempts to find a model that fits the data. An example of such an algorithm is the decision tree, which iteratively partitions the feature space to separate points with different labels. In Section 3.3, the neural network will be introduced, a different architecture for classification that learns features from the data itself.

3.2 Sequence labeling

In the digit recognition problem described in the previous section, the assumption is made that all training examples are independently and identically drawn from the joint distribution $P(x, y)$. However, in many real world problems the training examples are not independent of each other. Each training sample is a multivariate sequence $x_1, \ldots, x_n$, which has to be assigned a sequence of labels $y_1, \ldots, y_m$, where $m \leq n$. The case $m = 1$ is known as sequence classification, other cases ($m > 1$) as segment classification. An example of sequence classification is single word identification, where a single word is recognized from an audio waveform. When attempting to recognize a sentence from an audio waveform, multiple labels have to be predicted. These labels are dependent on one another, which means sequence classification can no longer be used, and segment classification is the appropriate method.

A possible solution method for the sequence labeling problem is the sliding window approach, where each training example is divided into time windows, to which a supervised classification algorithm is then applied. This approach is often applied when the input data is a high frequency signal, where the individual samples of a signal have less information than features extracted from a window of signals. The downside to the sliding window approach is that it does not take correlations of nearby labels into approach.

A popular approach to sequence labeling is the HMM [32], which is defined by two distributions: an observation distribution $P(x|y)$ which models how the observations depend on the labels, and the transition distribution $P(y_t|y_{t-1})$ which models how adjacent labels are related. The observation and transition matrix can be learned from the data, and the most likely label sequence is found with a dynamic programming algorithm. HMMs have been successful in a wide range of applications, for example speech recognition [12] or gesture recognition [35]. The discriminative analogue to HMMs are CRFs, which model the conditional (labels given features) instead of the joint distribution of the labels and features. CRFs have been successful for parsing [33] and gesture recognition [40]. The current best results in sequence labeling are achieved with RNNs, which will be introduced in section 3.5.

3.3 Neural networks

The field of machine learning is concerned with enabling computers to learn, and has seen a huge rise in popularity in the last decades. Different models have been suggested to enable a computer to learn, varying in complexity and success. One class of models, Artificial Neural Networks (ANNs), is based on the inner workings of the brain, which is unsurprising considering that the brain is also the thing that came up with the idea of machine learning. The human brain is made up of neurons, which transmit information through the brain, which is essential to our functioning. The information is used, for example, to notify us of pain, to make us perform complex movements, or to understand what we are seeing. There are about 100 billion neurons in the human brain, which are all connected to other neurons. An ANN is also built out of multiple layers of neurons, which are more commonly called units in ANN terminology (the terms neuron and unit will be used interchangeably in the remainder of this thesis).

The first layer is the input layer, where new inputs are presented. This input is passed through one or more *hidden layers*, and finally to the output layer. An ANN represents a function from output to input, and a network with just a single hidden layer has been shown to be a *universal function approximator*: it can approximate any continuous function on a compact input domain to arbitrary precision [17], as long as no restraint is placed on the number of hidden units.

Consider a network with n input units, with input vector $x = (x_1, x_2, \ldots, x_n)$. Each hidden layer takes as input the vector of outputs h^{l-1} from the previous layer (this can be the input layer, in this case $h^{l-1} = x$), first applies an affine transformation by weight matrix w^l and bias vector b^l to calculate the weighted input z^l. Next, a non-linear activation function f is applied to arrive at the intermediate output vector h^l:

$$h^l = f(z^l) = f(w^l h^{l-1} + b^l).$$

The activation function is usually non-linear; a neural network with just linear activation functions would just be a linear classifier. Popular activation functions are the the logistic sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

and the hyperbolic tangent

$$\tanh x = \frac{e^{2x} - 1}{e^{2x} + 1}.$$

The two functions are related by a linear transformation:

$$\tanh x = 2\sigma(2x) - 1.$$

A third popular activation function, which has been gaining popularity, is the Rectified Linear Unit (ReLU), which is defined as

$$\mathrm{ReLU}(x) = max(0, x)$$

In Figure 3.2, the three activation functions are plotted. The reasons for choosing a particular activation function will be explained in Section 3.4.

The output layer works like a hidden layer, where the choice of activation function depends on the type of task. For classification, it is common to use the *softmax* function, which squashes an m-dimensional vector of arbitrary real values into the an

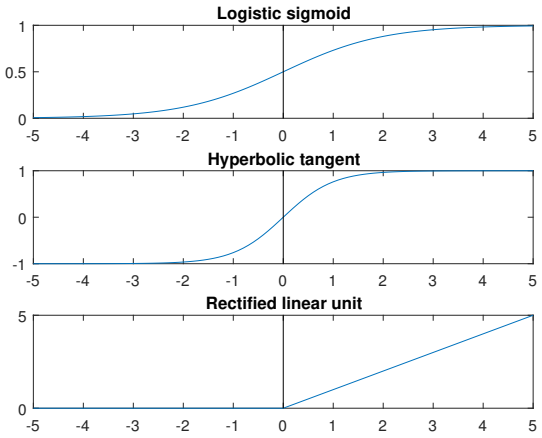

FIGURE 3.2: Three popular activation functions

m-dimensional vector of values in the interval (0,1), that sum to 1:

$$\text{softmax}(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{m} e^{x_k}} \text{ for } j = 1, \ldots, m$$

The j-th component of this vector is the estimated probability that x belongs to class j.

3.4 Learning in neural networks

The quality of a neural network is largely dependent on the values of the weights, and this is also where the power of neural networks comes in. Choosing an appropriate set of weights for a (non-trivial) problem is almost impossible to do by hand, and unnecessary. Instead, we let the neural network learn the weights by itself, using the method of *backpropagation*. To make this work, a cost function $C(w, b)$ has to be defined, that defines how good the output of the network is.

3.4.1 Cost function

A popular cost function in many machine learning approaches is the mean squared error (MSE), which, for a network with L layers and a dataset of n training examples, is defined as follows:

$$C(w, b) = \frac{1}{2n} \sum_{x} (y(x) - h^L)^2,$$

where $y(x)$ denotes the label for training example x, h^L is the output of the last layer of the network and the sum is taken over all x in the training set. The cost is a function of w and b because h^L depends on those parameters.

Learning in neural networks scales with the gradient of the cost, which can lead to problems when using MSE. To analyze the behavior of MSE, consider a single neuron,

where $h = f(z) = f(wx + b)$. The partial derivative of the cost with respect to the weights is given by

$$\begin{aligned}\frac{\partial C}{\partial w} &= \frac{1}{n}\sum_x (h - y(x))\frac{\partial h}{\partial w} \\ &= \frac{1}{n}\sum_x (f(z) - y(x))f'(z)\frac{\partial z}{\partial w} \\ &= \frac{1}{n}\sum_x (f(z) - y(x))f'(z)x,\end{aligned}$$

where the sum is taken over all n training examples. This includes the derivative of the activation function, which can get small if a neuron's output is saturated (its output is close to 0 or 1), which slows down learning. A different cost function which does not suffer from this behavior is *cross-entropy cost*, which is defined for a single neuron as:

$$C(w, b) = -\frac{1}{n}\sum_x [y \ln h + (1 - y)\ln(1 - h)].$$

For a single neuron, the partial derivative of the cost function with respect to the weights is:

$$\begin{aligned}\frac{\partial C}{\partial w} &= -\frac{1}{n}\sum_x \left(\frac{y(x)}{h} - \frac{1 - y(x)}{1 - h}\right)\frac{\partial f}{\partial w} \\ &= -\frac{1}{n}\sum_x \left(\frac{y(x)}{f(z)} - \frac{1 - y(x)}{1 - f(z)}\right)\frac{\partial f}{\partial w} \\ &= -\frac{1}{n}\sum_x \left(\frac{y(x)}{f(z)} - \frac{1 - y(x)}{1 - f(z)}\right)f'(z)x\end{aligned}$$

When using the sigmoid activation, the convenient property of the derivative $\sigma'(x) = \sigma(x)(1 - \sigma(x))$ causes the derivative of the cost function to become

$$\frac{\partial C}{\partial w} = -\frac{1}{n}\sum_x x(\sigma(x) - y).$$

In general, cross entropy loss is therefore preferred as a cost function over the MSE.

3.4.2 Gradient descent

The goal is to minimize the cost function, which can be effectively done with the method of *gradient descent*, which finds a local minimum of a function by starting at an initial point $\theta_1 = (w, b)$ for some weights w and biases b, and iteratively moving in the opposite direction of the gradient:

$$\theta_{k+1} = \theta_k - \eta \nabla C(\theta_k),$$

where η is the learning rate, which defines the magnitude of the move in this direction.

Performing gradient descent on the whole training set (batch gradient descent) can be time consuming, as the gradients for the whole training set need to be computed before performing a weight update, and it also doesn't allow for on line updating (where new examples are added during the gradient descent). An often used alternative to Batch Gradient Descent is *Stochastic Gradient Descent* (SGD), where the weights are updated for every training example $(x^{(i)}, y^{(i)})$:

$$\theta_{k+1} = \theta_k - \eta \nabla C(\theta; x^{(i)}; y^{(i)}).$$

SGD is usually much faster and can be used for on line updates. While these benefits make it preferable above Batch Gradient Descent, it does have a higher fluctuation, and makes it sensible to overshooting a local minimum. The benefits of both methods are combined in mini-batch gradient descent, which performs the weight update in mini-batches of l training examples:

$$\theta_{k+1} = \theta_k - \eta \nabla C(\theta; x^{(i:i+l)}; y^{(i:i+)}),$$

where i is the number of *epochs*, or training runs. This method can lead to more stable convergence, and can make use of efficient matrix-vector operations that are available in many scientific computing libraries.

Even with these improvements, the learning rate remains a problematic part of gradient descent: if it is too low, convergence can be slow; if it is too high, the algorithm could possibly diverge. A proposed solution is to use simulated annealing: reducing the learning rate during the learning process. In the early stage of the learning procedure, the parameter space can be 'scanned', and in the later stages, convergence can be achieved. The parameters for annealing have to be set in advance, and hence cannot adapt to the dataset. Recently, adaptive gradient descent methods have been proposed, which compute adaptive learning rates, such as Adaptive Moment Estimation (ADAM). A full description and comparison of these algorithms can be found in [31].

3.4.3 Backpropagation

The gradient of the cost function is calculated using the backpropagation algorithm. First, the neural network is initialized with weights w and biases b, which are usually randomly sampled from a normal distribution with mean zero and a low standard deviation. After initialization, a *forward pass* is performed, in which the vectors z^l and h^l are computed for each layer. After this, the error δ^L (in the output layer) is computed:

$$\delta^L = \nabla_h C \odot f'(z^L),$$

where $\odot$ denotes element-wise multiplication. The intuition behind this is as follows: the gradient $\nabla_h C$ measures how fast the cost is changing as a function of each output activation; if the cost does not depend much on a unit, this will have a negative effect

on δ^L. The second term measures the rate of change of the activation function at z^L. The error is back-propagated through the layers in the *backward pass*:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot f'(z^l), \tag{3.1}$$

where $(W^{l+1})^T$ is the transposed weight matrix of layer $l+1$. With this information, the gradient of the cost function can be computed; the partial derivatives for the biases and weights become:

$$\frac{\partial C}{\partial b_i} = \delta_i \quad \text{and} \quad \frac{\partial C}{\partial w_{jk}} = h_k \delta_j,$$ [1]

where δ_i is evaluated at the same unit as bias b_i, and h_k and δ_j respectively denote the activation from the unit input to the weight w_{jk} and the error of the unit output from the weight w_{jk}.

The choice for sigmoid, tanh or ReLU activation functions is convenient because their derivatives are easy to compute:

$$\begin{aligned} \sigma'(x) &= \sigma(x)(1-\sigma(x)) \\ tanh'(x) &= 1 - tanh^2(x) \\ \text{ReLU}'(x) &= \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

3.4.4 Regularization techniques

Machine learning techniques are always prone to overfitting, and neural networks are no different in this aspect. *Regularization techniques* can be used to avoid this phenomenon. A popular regularization technique is L2 regularization, or *weight decay*. This involves adding an extra term to the loss function, to penalize weights:

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2,$$

where C_0 is the original cost function (e.g. cross entropy), and λ is the regularization parameter ($\lambda > 0$). This makes sure that the network prefers to learns small weights, while also attempting to minimize the original cost. The relative importance of the two objectives depends on λ. Unregularized neural networks can learn complex models that do not generalize well, where regularized models are simpler models that do. Very similar to L2 regularization is L1 regularization, where the absolute value of the weights (instead of the square) is added to the loss:

$$C = C_0 + \frac{\lambda}{n} \sum_w |w|,$$

A different regularization technique, which is designed specifically for neural networks, is *dropout*. Instead of modifying the cost function, the neural network itself is modified during backpropagation. Before each forward pass, a randomly sampled fraction of

[1] A complete derivation of the backpropagation equations is found in [6].

the nodes and their connecting arcs are temporarily left out of the network. The probability that nodes are kept is the *keep probability* α. The input is propagated forward and back through the modified network for a batch of training examples, after which the appropriate weights and biases are updated. For each new batch, different nodes of the network are left out; the size of the fraction remains the same for each batch. Dropout was originally proposed in [15], and the technique removes the reliance of neurons on specific other neurons, as neurons are left out with a certain probability. This stimulates learning more robust features which are useful together with many different subsets of neurons.

3.4.5 Unstable gradients in deep neural networks

A common problem that occurs when training deep neural networks (with more than one hidden layer) is an unstable gradient. The gradient contains a product of the weights matrix in a layer and the derivative of the activation function (cf. equation 3.1). In the case of an n-layer neural network, the gradient of the shallowest hidden layer (closest to the input layer) contains this product n times, one layer deeper contains this product $n-1$ times, and so on.

The sigmoid and tanh activation functions have the property that their derivatives are small, especially when neurons reach saturation. As a consequence, shallow layers learn a lot slower than deeper layers, and can even cause the entire network to get stuck. This is known as the *vanishing gradient problem*. Another possibility is that weights grow during training, which causes the eigenvalues of the weight matrix to become large, and the gradient will grow exponentially in earlier layers, which is known as the *exploding gradient problem*. Both problems are described in more detail in [5]. The ReLU activation function does not suffer from this problem, but does have the property that its derivative is zero when the function is smaller than zero. This causes neurons to have zeros as output values, which leads to a sparse network. This has additional benefits, such as the fact that a sparse network is more robust to small changes in the input, as described in [14].

3.5 Recurrent Neural networks

Neural networks have been very successful in a variety of tasks, where they have outperformed many other machine learning methods. However, the traditional neural network architecture described in Section 3.3, where inputs are passed through a number of hidden layers, has no way to use information about past predictions in future ones. For sequence labeling, this would be a useful property, with a multitude of applications, for example in handwriting recognition; in Figure 3.3, the highlighted letter 'n' is ambiguous, but with the remaining letters, the word 'defence' is easily recognized.

Recurrent Neural Networks (RNNs) are a different neural network architecture, that allow information to persist, by allowing loops in the network. In a traditional neural network, information flows from the input layer, through the hidden layers, to the output layers. In an RNN, each hidden layer also receives information from the hidden layer in the previous timestep. Figure 3.4 depicts the architecture of an RNN, where the compact representation with a loop is unrolled into a representation that shows

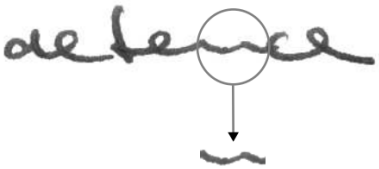

FIGURE 3.3: Importance of context for handwriting recognition. In isolation, the letter 'n' is ambiguous, but with the remaining letters, the word 'defence' is easily recognized

the RNN is just a series of copies of the same RNN, each one passing information to its successor.

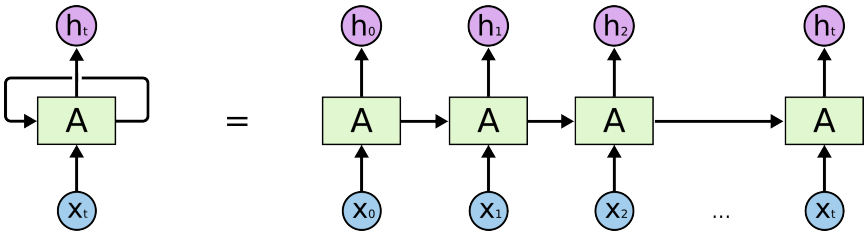

FIGURE 3.4: The recurrent neural network architecture, rolled and unrolled. Figure from [26].

The RNN receives a sequence of inputs $(x_1, \ldots, x_T)$, and computes a sequence of outputs $(y_1, \ldots, y_T)$. The output of hidden layer l at timestep t is computed with the following formula:

$$h_t^l = f(w^l h_t^{l-1} + w^t h_{t-1}^l + b^l).$$

where w^l and w^t are weight matrices connecting the current unit, respectively with the (possibly hidden) unit in the previous layer and the hidden unit at the previous timestep.

3.6 Long Short Term Memory

The backpropagation algorithm can be adapted to gradient calculation for an RNN, which is known as backpropagation through time (BPTT). The error is propagated not only through the hidden layers, but also through time steps. In BPTT, the unstable gradient problem becomes very relevant, as the number of multiplications in the gradient calculation can become very large. To overcome this, Hochreiter and Schmidhuber [16] introduced an alternative network architecture: the Long Short Term Memory (LSTM). Figure 3.5 shows part of an LSTM network, with focus on one LSTM *cell*.

The cell concept makes the LSTM such a powerful architecture, and much better suited for learning long term dependencies than a traditional RNN. Each cell has a state C_t, and multiple *gates* that decide the information that gets added or removed from the cell state. First of those is the forget gate, which decides which pieces of information are valuable to remember, and which pieces are not. The gate receives inputs x_t and h_{t-1}, and pushes them through a sigmoid layer, arriving at output f_t:

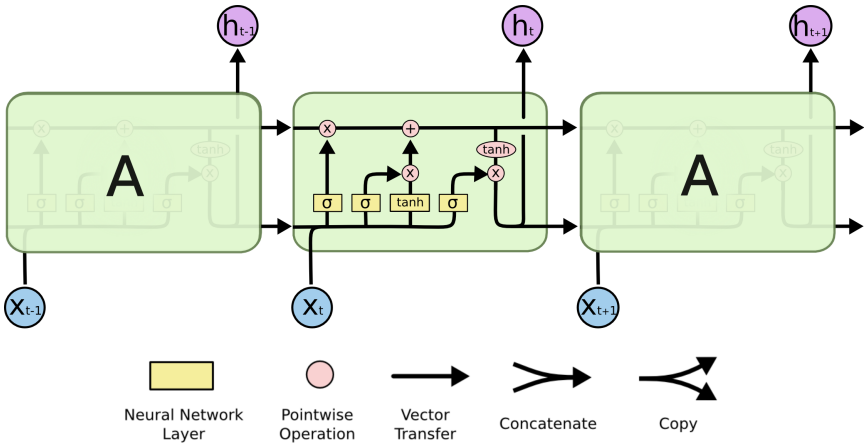

FIGURE 3.5: The LSTM architecture visualized. Figure from [26].

$$f_t = \sigma(w_f \cdot [h_{t-1}, x_t] + b_f),$$

where w_f and b_f are the weights and biases of this gate, and work on the concatenation $[h_{t-1}, x_t]$ of the hidden unit of the previous time-step, and the current input. Next, a vector of new candidate values $\tilde{C}$ is created

$$\tilde{C}_t = \sigma(w_C \cdot [h_{t-1}, x_t] + b_C).$$

Before adding these candidate values to the memory, the input gate learns which information is worth saving into the new cell state:

$$i_t = \sigma(w_i \cdot [h_{t-1}, x_t] + b_i).$$

With the old cell state and the forget gate, combined with the candidate values and the input gate, the cell state can be updated:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t.$$

The cell state can be seen as the Long Term Memory of the LSTM. The last step is to decide which part of the cell state is relevant to output. This is decided by the output gate:

$$o_t = \sigma(w_o \cdot [h_{t-1}, x_t] + b_o),$$

which is applied to the new cell state, after it has been passed through a tanh activation, resulting in the new hidden output:

$$h_t = o_t \odot \tanh(C_t).$$

3.6.1 Variations on LSTM

The LSTM is the most popular and widely used variant on the RNN, but many different architectures have been proposed. One alternative is the Gated Recurrent Unit (GRU), which combines the forget and input gates into a single update gate z_t, has a reset gate r_t and combines cell state and hidden state, to arrive at a simpler model:

$$\begin{aligned} z_t &= \sigma(w_z \cdot [h_{t-1}, x_t]) \\ r_t &= \sigma(w_r \cdot [h_{t-1}, x_t]) \\ \tilde{h}_t &= \tanh(w_{\tilde{h}} \cdot [r_t \odot h_{t-1}, x_t]) \\ h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \end{aligned}$$

In a 2015 paper, Dosovitskiy, Springenberg, and Brox [11] conduct an empirical exploration of over 10,000 RNN architectures, where they failed to find any architecture that consistently outperformed the LSTM or GRU architecture.

Chapter 4

Experiments

The classification of posture and activity is a supervised learning problem, and an algorithm to achieve this task needs labeled training data. Section 4.1 describes the data collection and processing process, followed by a first attempt at the classification task, with a threshold based method, in Section 4.2. Sections 4.3 and 4.4 focus respectively on the detection and classification of transitions. Lastly, section 4.5 outlines a method to classify postures, activities and transitions.

4.1 Data collection and processing

Data was collected with a wearable device, with an embedded InvenSense MPU-6000 triaxial accelerometer. Figure 4.1 shows a schematic representation, with the orientation of the accelerometer axes.

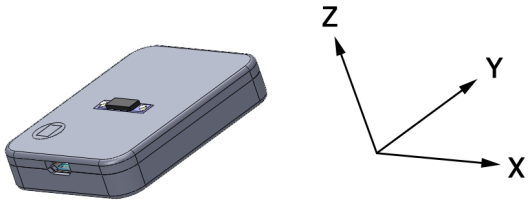

FIGURE 4.1: The wearable device and orientation of the accelerometer axes.

The data is collected at a rate of 31.25 Hz, which has been proven sufficient to assess physical activity [7]. A 6 minute protocol was performed by 30 participants, with one device around the arm and one on the chest. The protocol consists of 5 postures, 3 activities and 6 transitions, which are listed in Table 4.1. The table also shows the total recorded time per posture, activity and transition, summed over all participants, with a total recording time of 180 minutes. For different participants, the order of the postures and activities has been changed, to ensure variation among the recordings. Figure 4.2 shows a participant during the protocol, during the sitting posture.

After collection of the data, each recording is annotated by hand with the corresponding labels. To remove noise, the acceleration data is filtered with a low pass Butterworth filter with a cutoff frequency of 20 Hz, as human motion is below this frequency. The resulting signal is separated into gravitational and body acceleration using another low pass Butterworth filter with a cutoff frequency of 0.3 Hz, using the

FIGURE 4.2: A participant during the protocol, equipped with a device on the arm and chest.

Label	**Description**	**Time (minutes)**	Category
LIE-BACK	Lying on back	11	posture
LIE-RIGHT	Lying on right side	12	posture
LIE-LEFT	Lying on left side	12	posture
SIT	Sitting on a chair	28	posture
STAND	Standing upright	26	posture
WALK	Walking at normal pace	24	activity
DOWNSTAIRS	Walking down the stairs	19	activity
UPSTAIRS	Walking up the stairs	21	activity
UPWARDS	Upwards transition	9	transition
DOWNWARDS	Downwards transition	8	transition
RIGHT-TURN	Turning right while lying down	4	transition
LEFT-TURN	Turning left while lying down	4	transition

TABLE 4.1: The different postures, activities and transitions in the protocol, and their total recorded time across all recordings

fact that gravitational acceleration has only low frequency components [2]. The result is a signal with 9 parameters: the 3 dimensional body acceleration, gravitational acceleration, and gyroscopic data.

Figure 4.3 shows a recording of one participant [1], where the first subplot is the body acceleration, the second is the gravitational acceleration and the third is the angular velocity measured by the gyroscope. Postures and activities are annotated in the first subplot. The body acceleration shows a clear distinction between posture and activity; there is much more accelerometer activity in the latter case. However, also postural segments show some accelerometer activity, which can arise due to minor movements of the user or external noise. The gravitational acceleration shows a clear distinction between different postures, as the accelerometer axes are oriented differently relative to the direction of the gravity vector in each posture.

[1] In this plot, and following plots, the values for acceleration and angular velocity are left out, as they are irrelevant and clutter plots.

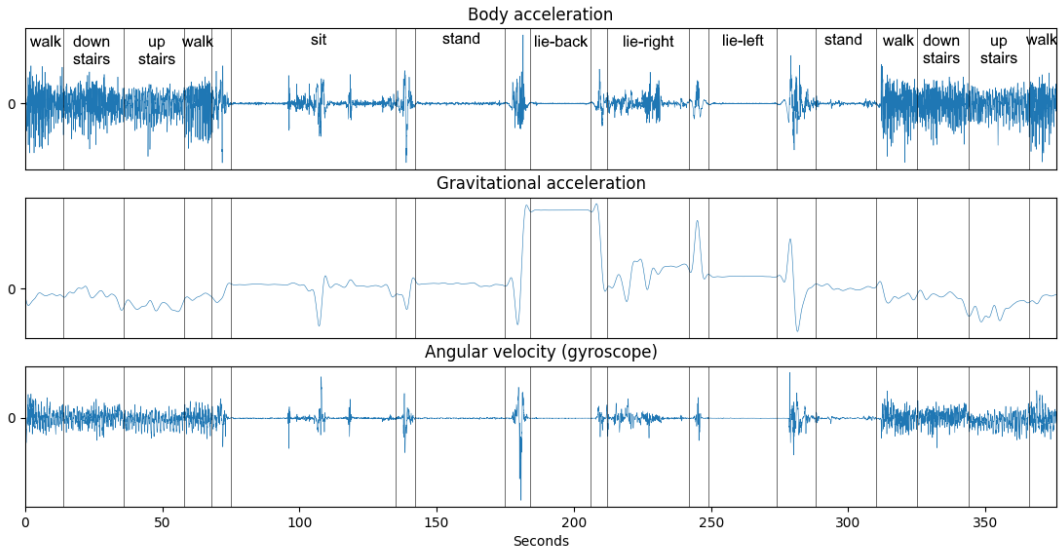

FIGURE 4.3: Signals from one recording; the first subplot is the body acceleration, the second is the gravitational acceleration and the third is the angular velocity, measured by the gyroscope.

4.2 Posture and activity classification

In this section, the aim is to develop a simple method that can detect and classify postures and activities. A sliding window is moved over the signal, and features are extracted from each window. In order to distinguish between active and static windows, the standard deviation σ of the acceleration is calculated. In Figure 4.4, a density plot is shown for σ, which shows a clear difference between the distributions of the two different types of windows (postural transitions are included in the active windows): the static windows have a very low σ, because there is very little movement, where the active windows have more movement, but are also more spread out, because not all active windows have the same amount of movement in them.

As a second feature, the mean of the gravitational acceleration is computed, for each of the 3 spatial components. The results for the different postures are plotted in Figure 4.5. The three lying postures (on back, on the left and right side) are well-separated from the upright postures (sitting and standing), and from each other. The upright postures are not separable in this space, which arises from the fact that the device is oriented the same way when sitting as it is while standing, and as such the gravity vector is oriented the same as well.

With these insights, a threshold based method is developed, that processes the signal with a sliding window. First, the acceleration standard deviation σ, is compared to a threshold t_1, to distinguish between activity and the postures. Then, the y-component of the gravitational acceleration is compared to a threshold t_2 to distinguish upright posture (sitting or standing) from lying down. Lastly, the z-component is compared to thresholds t_3 and t_4 to differentiate between lying on different sides. The flowchart for this threshold based algorithm is shown in Figure 4.6.

The algorithm is tested for window lengths of 1, 2, 4, 8 and 16 seconds; the 8 second window obtained the best classification result. The resulting confusion matrix is

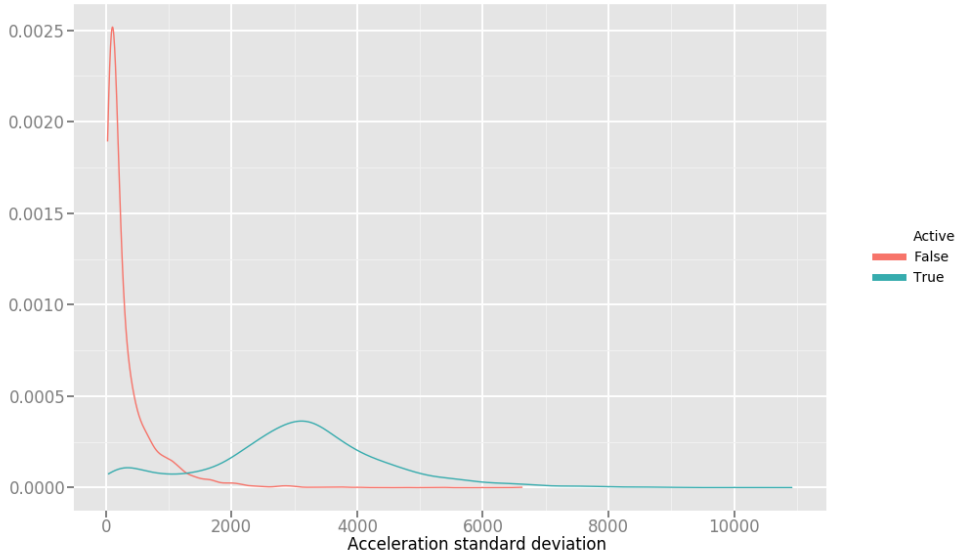

FIGURE 4.4: A density plot of the standard deviation of the acceleration, for active (blue) and static (red) windows

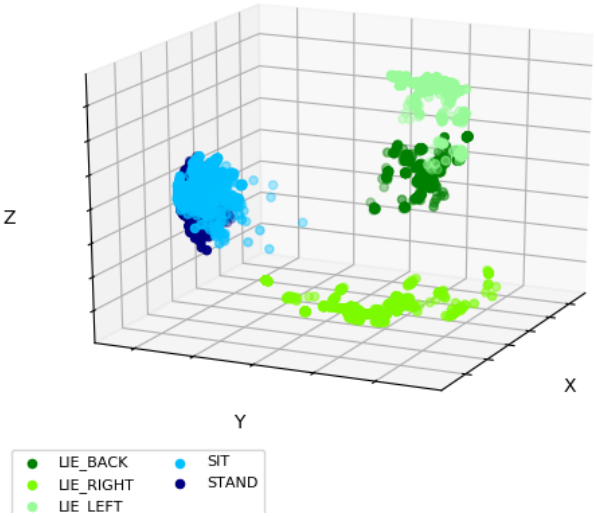

FIGURE 4.5: The 3 components of gravitational acceleration for the different postures

shown in Figure 4.2, where rows correspond to the actual labels, and columns to the predictions made by the algorithm. The accuracy of this method is 91.8%. Misclassifications mostly arise when activity/transition windows get labeled as postures, or vice versa. This arises partly from noise, and partly from the fact that the labeling is done by hand, and thus can not be perfect either. The power of this method is its simplicity, which makes it very efficient and easy to port to a wearable device. However, the method is not very sophisticated, as it can not distinguish between different

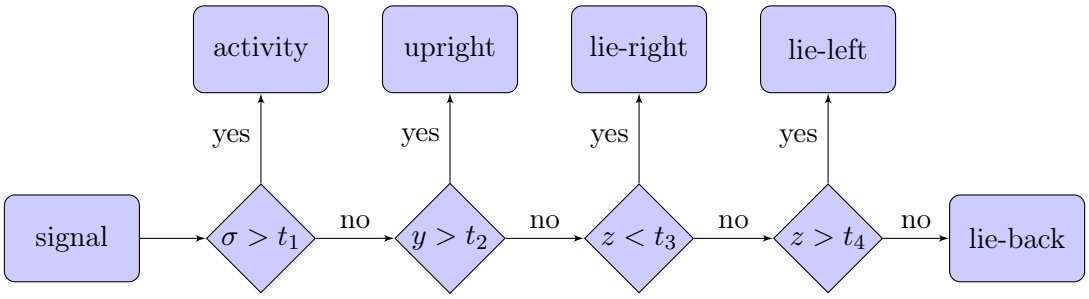

FIGURE 4.6: The threshold based classification algorithm

activities, and does not take transitions into account.

	LIE-BACK	LIE-RIGHT	LIE-LEFT	UPRIGHT	ACTIVITY
LIE-BACK	**517**	0	0	22	55
LIE-RIGHT	18	**581**	0	7	51
LIE-LEFT	97	0	**517**	0	47
UPRIGHT	0	0	0	**2666**	299
ACTIVITY	53	33	30	59	**4400**

TABLE 4.2: Confusion matrix for the threshold based method. Rows correspond to the actual labels, and columns to the predictions made by the algorithm

4.3 Transitions detection

Postural transitions play an important part in activity monitoring; and provide important information about the health of a user, e.g. when combined with heart rate or blood pressure. Anomalous changes in heart rate or blood pressure when a patient is standing up or lying down can be indicators of heart problems, and are important to be detected. Transitions from one position to another when lying down can also be valuable, especially when a patient suffers from apnea, which usually occurs less when sleeping on the side than when sleeping on the back.

Postural transitions are recognizable by a change in gravitational acceleration, as can be seen in Figure 4.3. To detect these changes, the minimum is subtracted from the maximum of the gravitational acceleration g in a window, and this is summed over the three dimensions x, y and z, to calculate the gravitational difference d:

$$d = \sum_{k \in \{x,y,z\}} \max(g_k) - \min(g_k)$$

A simple peakfinder algorithm [25] can be used to find the changes in gravitational acceleration, which may indicate a transition. The algorithm has a parameter $\alpha \in [0, 1]$, which indicates the threshold to detect peaks (relative to the highest peak in the signal). Figure 4.7 shows the gravitational difference of an example recording, and detected peaks for $\alpha = 0.5$ and $\alpha = 0.2$.

To score the detection algorithm, the binary confusion matrix is defined in Figure 4.3. Like before, rows correspond to the true labels, and columns correspond to the

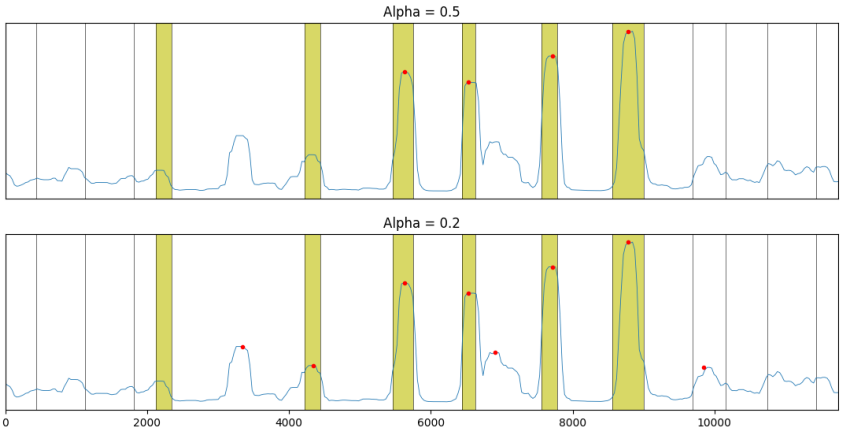

FIGURE 4.7: Transition detection for different values of the threshold α. Shaded yellow segments are transition segments, red dots are the detected peaks in gravitational acceleration.

		Predicted t	**Predicted** $\bar{t}$
True t		true positive	false negative
True $\bar{t}$		false positive	true negative

TABLE 4.3: Binary confusion matrix for transition detection

predicted labels, where t denotes a transition and $\bar{t}$ denotes a non-transition. Four cases exist:

1. a transition is correctly detected: true positive (tp),
2. a transition is falsely detected: false positive (fp),
3. a transition is not detected: false negative (fn)
4. no transition is detected when there is none: true negative (tn)

A lower value of α detects more peaks, but also has a higher number of false positives. To formalize this, define precision and recall as:

$$\text{precision} = \frac{\text{tp}}{\text{tp} + \text{fp}} \quad \text{and} \quad \text{recall} = \frac{\text{tp}}{\text{tp} + \text{fn}},$$

Precision and recall are calculated for different values of α, the result is plotted in Figure 4.8. There is not one optimal value of α; this depends on the relative importance

of precision and recall for the application. If both are considered equally important, the optimum is found at $\alpha = 0.22$, with a precision of 0.85 and a recall of 0.86.

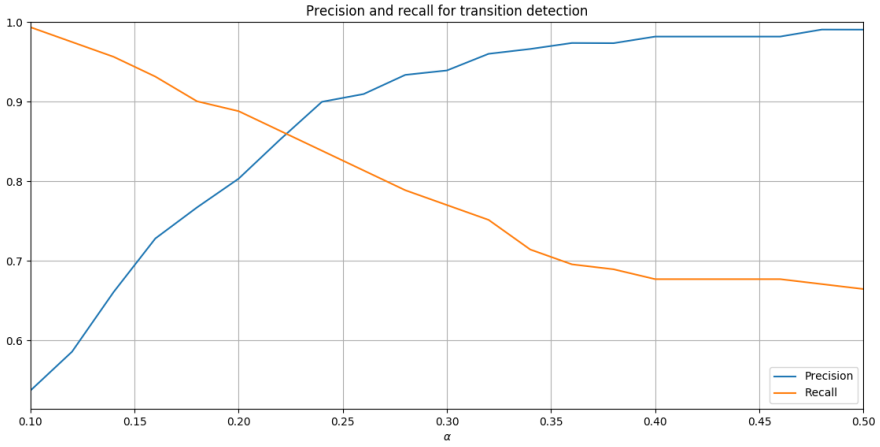

FIGURE 4.8: Precision and recall values for different values of α

4.4 Transition classification

The transition detection algorithm is only valuable when the detected transitions are also classified into one of the transition classes. Because of the occurrence of false positives in the transition detection algorithm, the transition classification problem has 5 classes: UPWARDS, DOWNWARDS, RIGHT-TURN, LEFT-TURN and NULL, where NULL indicates the occurrence of a transition during a non-transition segment. An 8 second time window is extracted from the signal, centered around each transition detected with the algorithm from Section 4.3. The signals in this time window are then classified into one of the five classes. A number of different machine learning methods are compared, to see which obtains the best result. Two different neural network architectures are tested:

1. a neural network with 3 fully connected hidden layers with ReLU activation and a fully connected output layer with softmax activation, which will be referred to as the non-recurrent architecture,
2. a neural network with 3 LSTM layers with ReLU activation and a fully connected output layer with softmax activation, which will be referred to as the recurrent architecture

For both architectures, different hyperparameters are tested, where hyperparameters are the parameters of the network that are chosen before optimizing the other parameters. Examples of hyperparameters are the number of hidden units, the learning rate and the regularization parameter. The other parameters in the network are the weights and the biases, which get optimized with gradient descent. In the hyperparameter search, the networks are trained with mini batch gradient descent, with a batch size of 10, where a new mini batch is randomly sampled from the training set every epoch, for a total of 500 epochs. The learning rate is 0.001, and the optimizer is chosen to be ADAM. The training set consists of 70 % of the transitions, and the testing set is the other 30 %.

An important hyperparameter is the number of hidden units. Networks with more units have more learning capacity, but also take more time to train and can be more prone to overfitting due to the high number of parameters. To see what the influence of the number of hidden units is on the prediction accuracy, 5 networks are trained for both architectures. The number of hidden units is chosen to be 10, 20, 50, 100 and 200. The accuracy on the training and testing set is shown in Figure 4.9. The number of hidden units has an influence on the achieved accuracy for the non-recurrent architecture; the networks with 10 or 20 hidden units are outperformed by the networks with more units. The recurrent architecture does not show such a clear distinction, but learns faster with less hidden units. The bigger networks do have a much higher accuracy on the training set than the testing set: they are overfitting. For the non-recurrent architecture, the networks with 50 and 100 units reach a 100 % accuracy on the training set. This means the cost function is equal to zero, the learning stops, and the accuracy on the test set remains constant. When the recurrent network is run for 1000 epochs, it also reaches 100 % train set accuracy and stops learning.

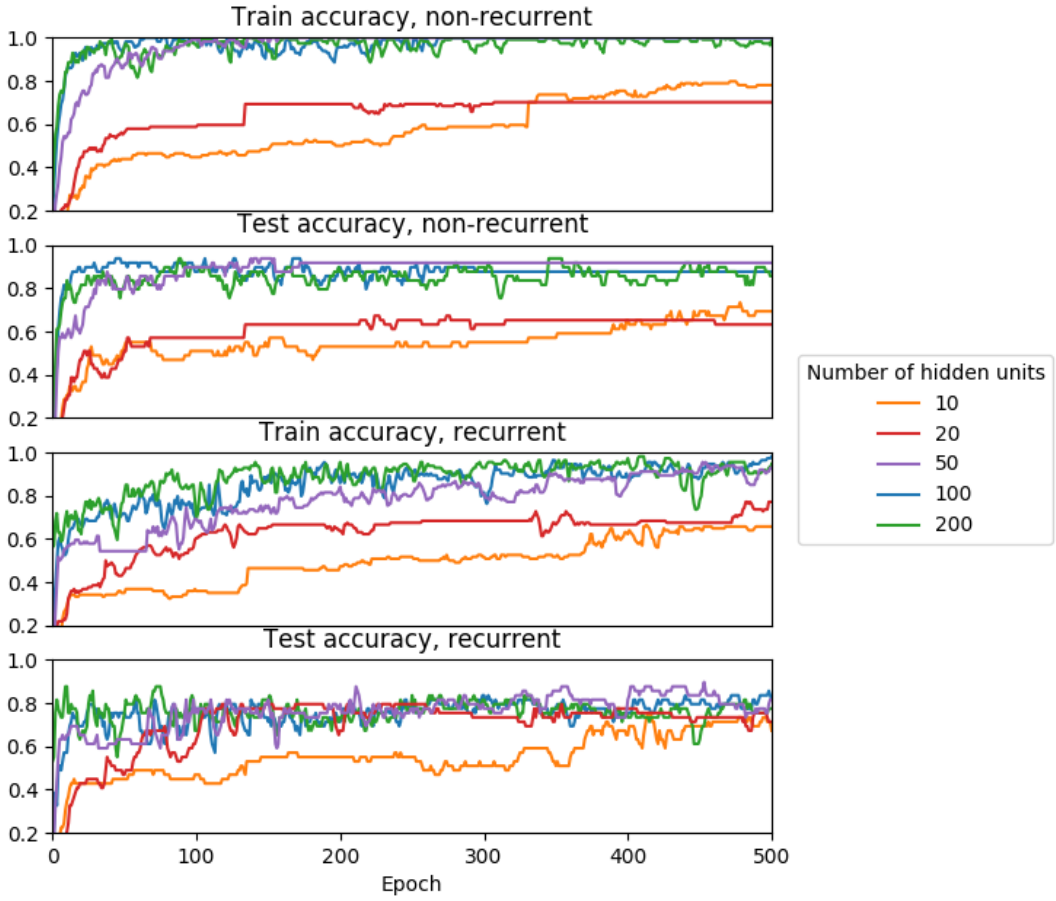

FIGURE 4.9: Classification accuracy on the train and test set for both neural network architectures, with different numbers of hidden units

This overfitting occurs because of the small size of the set of transitions: only 163 transitions are detected in all the recordings. To improve the generalization of the model, two regularization methods are tested: L2 regularization and dropout, as described in 3.4.4. A grid search is performed over the regularization parameter λ and the keep probability α, with $\lambda \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\, 1\}$ and $\alpha \in \{1, 0.8, 0.6, 0.4, 0.2\}$ for the non-recurrent network. For the recurrent network, the grid search is performed for

$\lambda \in \{10^{-3}, 10^{-2}, 10^{-1}\}$ and $\alpha \in \{1, 0.8, 0.6\}$, as the networks take considerably longer to train than their non-recurrent counterparts. The mean accuracy over the last 100 epochs is calculated and plotted as a function of λ and α in Figure 4.10

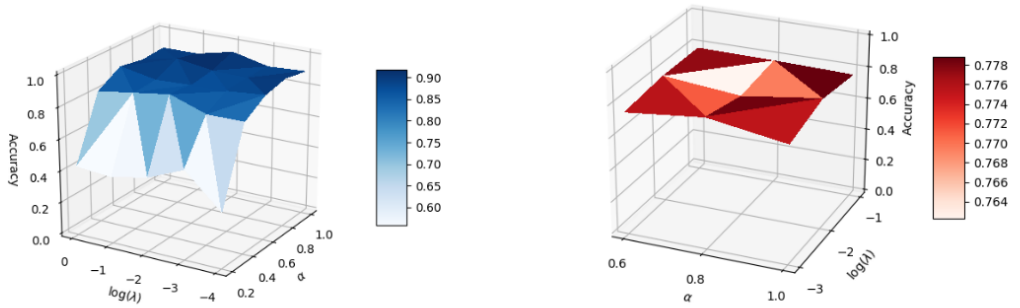

FIGURE 4.10: Surface plots of the accuracy of the transition classification task as function of λ and α. Left the non-recurrent network, right the recurrent network

For the non-recurrent network, the plot hows some interesting results: for low values of α, the regularization has a negative effect, because too little of the network is kept to learn. In general, dropout does not seem to have a positive effect on the classification accuracy, where the l2 regularization does have a positive impact. The optimal accuracy is 0.94, with $\lambda = 10^{-2}$ and $\alpha = 1$ (which means dropout is not used, as α is the keep probability) with an accuracy of 0.88 for the non-regularized network.

The recurrent network reaches the optimum value at the exact same values of α and lambda, but only has a test set accuracy of 80 %. This probably has to do with the fact that the recurrent network has many more parameters than its non-recurrent counterpart; each LSTM cell features multiple weight matrices, where a fully connected layer has just one. This makes the recurrent network even more prone to overfitting than a non-recurrent one, which might explain the accuracy decrease.

Method	Accuracy
ANN	0.88
RNN	0.76
ANN + L2	**0.94**
RNN + L2	0.80
Naive Bayes	0.79
Decision Tree	0.80
Random Forest	0.88

TABLE 4.4: Test set accuracy of different methods for the transition classification task. ANN denotes the non-recurrent neural architecture, L2 denotes L2 regularization

As a comparison, the transitions are also classified with three other machine learning methods. For these methods, features are extracted from the signal for each of the 9 parameters: mean, standard deviation, minimum, maximum, argument of the minimum and the argument of the maximum. First, Naive Bayes is used, a method that is

very simple but works surprisingly well for many applications. Next, a decision tree is fitted to the data, which recursively partitions the parameter space (both methods are described in [6]). Lastly, a random forest (an ensemble method using decision trees [37]) with 500 estimators is fitted to the data. Naive Bayes has an accuracy of 0.79, where the decision tree and random forest respectively reach 0.80 and 0.88. Table 4.4 summarizes the results of the transition classification.

4.5 Posture, activity and transition classification

The threshold based method from Section 4.2 works well, but only classifies each window in one of five classes. Ideally, a finer grained classification is made into all 12 classes listed in table 4.1, including the postural transitions. In this section, the same methods from Section 4.4 are applied to classify postures, activities and transitions. First, the same sliding window as in Section 4.2 is used to extract time windows of 8 seconds from the signal. These windows are then classified individually.

The training set is much larger this time: a total of 9452 time windows are to be classified. This means the neural network will take much longer to learn. For the non-recurrent architecture, a grid search of α and λ is performed. The network is trained for 10000 epochs, with a batch size of 10. For time reasons, a smaller grid is searched this time: $\alpha \in \{0.6, 0.8, 1\}$ and $\lambda \in \{-3, -2, -1\}$. The resulting accuracies are plotted in Figure 4.11, where the 'accuracy'-axis has lower limit 0.5, to better visualize the result. The best test set accuracy is 0.73 and occurs at the same parameter values as in Section 4.4: $\lambda = 10^{-2}$ and $\alpha = 1$. Accuracy on the train set reaches 0.90 for these values, which means there is still overfitting taking place. Dropout seems to only negatively affect the accuracy, as was also the case with transition classification.

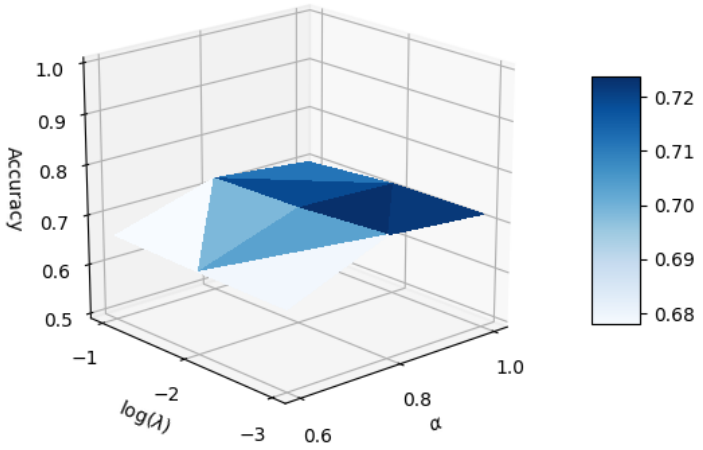

FIGURE 4.11: Surface plot of the accuracy on the window classification task as function of λ and α for th non-recurrent network.

The recurrent architecture for this classification task takes even longer to fully train, and is only tested for the optimal parameter values $\lambda = 10^{-2}$ and $\alpha = 1$. The

classification accuracy on the test set is 0.63, where train set accuracy is 0.83, which means this network is also overfitting. The Naive Bayesian and decision tree classifiers reach a higher accuracy with 0.70, and the best classification accuracy is 0.76 for the random forest.

4.5.1 Hybrid method

The window-based classifiers take a fixed sequence (the windowed signal) as input, and output a probability distribution over the labels. A sample output is plotted in Figure 4.12, where the different colors correspond to different classes. This distribution is then used to predict the label on the window, by taking the most likely label. The methods have so far just used the information in the window itself, and not the information of other windows that are nearby in the signal.

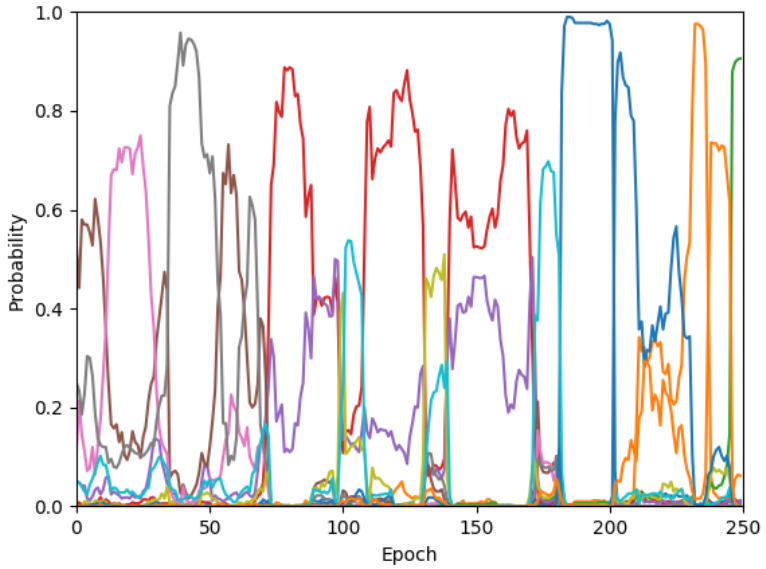

FIGURE 4.12: Output probabilities of the window classification task for part of a recording. Different colors correspond to different classes.

To incorporate this information, a second recurrent neural network is trained. As input for this network, the output from the random forest classifier is chosen, because it has the highest classification accuracy on the window classification task. This input consists of probability distributions over the labels for a sequence of windows, the output is a sequence of predicted labels for these windows. The network is trained with 3 fold cross validation, during 500 epochs, where in each epoch the training set is fed into the network. The accuracy of this hybrid method is 0.86, a major improvement over any of the individual methods. Table 4.6 shows the resulting confusion matrix. Most misclassifications occur when a transition is detected when there is none (false positive), or when a transition is not detected (false negative), a problem that was already apparent in Section 4.3. Table 4.5 summarizes the results of the window classification task.

Method	Accuracy
ANN + L2	0.73
RNN + L2	0.63
Naive Bayes	0.70
Decision Tree	0.70
Random Forest	0.76
Random Forest + RNN hybrid	0.86

TABLE 4.5: Test set accuracy of different methods for the window classification task. ANN denotes the non-recurrent neural architecture, L2 denotes L2 regularization

	LIE-BACK	LIE-RIGHT	LIE-LEFT	SIT	STAND	WALK	DOWNSTAIRS	UPSTAIRS	UPWARDS	DOWNWARDS	RIGHT-TURN	LEFT-TURN
LIE-BACK	**513**	0	46	0	0	3	0	4	0	24	4	0
LIE-RIGHT	0	**627**	0	0	0	0	0	6	4	0	15	5
LIE-LEFT	10	0	**635**	0	0	0	0	4	0	0	8	4
SIT	0	0	0	**1476**	0	15	12	14	14	14	0	0
STAND	0	0	0	33	**1248**	17	1	12	98	11	0	0
WALK	0	0	0	0	20	**759**	27	121	0	8	0	0
DOWNSTAIRS	0	0	0	0	0	71	**926**	47	0	4	0	0
UPSTAIRS	0	0	0	0	0	32	44	**1081**	0	0	0	0
UPWARDS	0	1	74	88	35	0	0	3	**274**	0	0	0
DOWNWARDS	9	0	0	29	63	70	3	2	0	**266**	0	0
RIGHT-TURN	41	2	12	0	0	0	0	0	0	0	**168**	0
LEFT-TURN	7	13	33	0	0	0	0	3	8	0	1	**178**

TABLE 4.6: Confusion matrix for the window classification task for the random forest + RNN combined method. Rows are the actual labels, columns the predictions.

Chapter 5

Conclusion

In Section 1.2, the **research question** of this thesis was posed:

Is it possible to accurately recognize posture and activity of an individual using accelerometer and gyroscope data acquired from a wearable device?

In addition, a sub-question was posed:

Do neural networks provide a better recognition of posture and activity, compared to other machine learning methods?

It is possible to distinguish between activity and postures, using the simple threshold based method from Section 4.2. This method has an classification accuracy of 0.92, but has the disadvantage that it can only distinguish between 5 different classes. The advantage of this method is its simplicity, which makes it easy to implement on the device.

The transitions detected with the algorithm from Section 4.3 can be accurately classified into one of the four transition classes, and the best accuracy is achieved with a regularized neural network. The LSTM network does not achieve a good classification accuracy, which might be caused by the small size of the training set, which leads to overfitting.

Finally, the posture, activity and transition classification task from Section 4.5 was best classified with a hybrid method, where the windows are assigned a probability distribution over the labels by a random forest classifier, and sequences of windows are then labeled with an LSTM network.

5.1 Discussion and future work

It is difficult to say which method is best, and this also depends on the application. The hybrid method achieves the best classification accuracy, but is also very computationally expensive. Simpler window based classifiers, such as Naive Bayes, achieve an acceptable accuracy and are easily integrated into the device. The transition detection and classification algorithms can easily be implemented as well.

The data collected for this thesis was limited, and a more extensive data collection could improve classification accuracy for all algorithms, and make the algorithms more robust to changes in input. More data would also mean that the neural networks are less prone to overfitting, a phenomenon that even the regularized networks still exhibit. More data could also extend the range of detected activities, such that more activities of daily living are recognized, such as cooking or cleaning.

The posture, activity and transition information becomes really interesting when combined with the other data that is collected on the device. A possible future research direction is to use the activity data together with heart rate data to give users a 'cardiac health score'. Another possibility would be to detect anomalous events in heart rate, to avoid heart failures. Along these lines, there are many applications, most of which have probably not been thought of yet!

Bibliography

[1] Felicity R Allen et al. "Classification of a known sequence of motions and postures from accelerometry data using adapted Gaussian mixture models." In: *Physiological measurement* 27.10 (2006), pp. 935–951. ISSN: 0967-3334. DOI: 10.1088/0967-3334/27/10/001.

[2] Davide Anguita et al. "A Public Domain Dataset for Human Activity Recognition Using Smartphones". In: *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning* April (2013), pp. 24–26. URL: http://www.i6doc.com/en/livre/?GCOI=28001100131010.

[3] Davide Anguita et al. "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7657 LNCS (2012), pp. 216–223. ISSN: 03029743. DOI: 10.1007/978-3-642-35395-6_30.

[4] Oresti Banos et al. "Window size impact in human activity recognition." In: *Sensors (Basel, Switzerland)* 14.4 (2014), pp. 6474–99. ISSN: 1424-8220. DOI: 10.3390/s140406474. URL: http://www.mdpi.com/1424-8220/14/4/6474/htm.

[5] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. *Learning Long-Term Dependencies with Gradient Descent is Difficult*. 1994. DOI: 10.1109/72.279181. arXiv: arXiv:1211.5063v2.

[6] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[7] Carlijn V C Bouten et al. "A triaxial accelerometer and portable data processing unit for the assessment of daily physical activity". In: *IEEE Transactions on Biomedical Engineering* 44.3 (1997), pp. 136–147. ISSN: 00189294. DOI: 10.1109/10.554760.

[8] Tomas Brezmes, Juan Luis Gorricho, and Josep Cotrina. "Activity recognition from accelerometer data on a mobile phone". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 5518 LNCS.PART 2 (2009), pp. 796–799. ISSN: 03029743. DOI: 10.1007/978-3-642-02481-8_120. URL: http://www.aaai.org/Papers/IAAI/2005/IAAI05-013.

[9] Li Deng. "The MNIST database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142. ISSN: 10535888. DOI: 10.1109/MSP.2012.2211477.

[10] Anind K. Dey and Gregory Abowd. "Towards a Better Understanding of Context and Context-Awareness". In: *Handheld and Ubiquitous Computing* 40.3 (1999), pp. 304–307. ISSN: 00219266. DOI: 10.1007/3-540-48157-5_29. URL: http://dx.doi.org/10.1007/3-540-48157-5{_}29.

[11] Alexey Dosovitskiy, Jost Tobias Springenberg, and Thomas Brox. "Learning to generate chairs with convolutional neural networks". In: *Proceedings of the IEEE*

Computer Society Conference on Computer Vision and Pattern Recognition 07-12-June (2015), pp. 1538–1546. ISSN: 10636919. DOI: 10.1109/CVPR.2015.7298761. arXiv: 1512.03385.

[12] Mark Gales and Steve Young. "The application of hidden Markov models in speech recognition". In: *Found. Trends Signal Process.* 1.3 (2007), pp. 195–304. ISSN: 1932-8346. DOI: 10.1561/2000000004. arXiv: arXiv:1011.1669v3. URL: http://dx.doi.org/10.1561/2000000004.

[13] Fernando García-García et al. "Statistical machine learning for automatic assessment of physical activity intensity using multi-axial accelerometry and heart rate". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6747 LNAI (2011), pp. 70–79. ISSN: 03029743. DOI: 10.1007/978-3-642-22218-4_9.

[14] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks". In: *AISTATS '11: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics* 15 (2011), pp. 315–323. ISSN: 15324435. DOI: 10.1.1.208.6449. arXiv: 1502.03167.

[15] Geoffrey E. Hinton et al. "Improving neural networks by preventing co-adaptation of feature detectors". In: *ArXiv e-prints* (2012), pp. 1–18. ISSN: 9781467394673. DOI: arXiv:1207.0580. arXiv: 1207.0580. URL: http://arxiv.org/abs/1207.0580.

[16] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. arXiv: 1206.2944. URL: http://www.mitpressjournals.org/doi/abs/10.1162/neco.1997.9.8.1735.

[17] Kurt Hornik, Maxwell Stinchcombe, and Halber White. "Multilayer Feedforward Networks are Universial Approximators". In: *Neural Networks* 2 (1989), pp. 359–366.

[18] Michael Hüsken and Peter Stagge. "Recurrent neural networks for time series classification". In: *Neurocomputing* 50.C (2003), pp. 223–235. ISSN: 09252312. DOI: 10.1016/S0925-2312(01)00706-8. URL: http://www.sciencedirect.com/science/article/pii/S0925231201007068.

[19] Sampath Jayalath and Nimsiri Abhayasinghe. "A gyroscopic data based pedometer algorithm". In: *Proceedings of the 8th International Conference on Computer Science and Education, ICCSE 2013* (2013), pp. 551–555. DOI: 10.1109/ICCSE.2013.6553971.

[20] Kari Laasonen et al. "Pervasive Computing". In: *Pervasive Computing* 3001 (2004), pp. 287–304. ISSN: 03029743. DOI: 10.1007/b96922. arXiv: 9780201398298. URL: http://www.springerlink.com/content/30wjfe9rxhmyn7up/.

[21] Qiang Li et al. "Accurate, Fast Fall Detection Using Posture and Context Information". In: *Sensys'08: Proceedings of the 6th Acm Conference on Embedded Networked Sensor Systems* (2008), pp. 443–444. DOI: 10.1109/p3644.45.

[22] G. M. Lyons et al. "A description of an accelerometer-based mobility monitoring technique". In: *Medical Engineering and Physics* 27.6 (2005), pp. 497–504. ISSN: 13504533. DOI: 10.1016/j.medengphy.2004.11.006.

[23] Andrea Mannini and Angelo Maria Sabatini. "Machine learning methods for classifying human physical activity from on-body accelerometers". In: *Sensors* 10.2 (2010), pp. 1154–1175. ISSN: 14248220. DOI: 10.3390/s100201154.

[24] M. J. Mathie et al. "Classification of basic daily movements using a triaxial accelerometer". In: *Medical and Biological Engineering and Computing* 42.5 (2004), pp. 679–687. ISSN: 01400118. DOI: 10.1007/BF02347551.

[25] Lucas Hermann Negri. *PeakUtils*. 2014. URL: http://pythonhosted.org/PeakUtils/index.html{\#}.

[26] Christopher Olah. *Understanding LSTM Networks*. 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[27] Francisco Javier Ordóñez and Daniel Roggen. "Deep convolutional and LSTM recurrent neural networks for multimodal wearable activity recognition". In: *Sensors (Switzerland)* 16.1 (2016). ISSN: 14248220. DOI: 10.3390/s16010115.

[28] Jun-geun Park et al. "Online pose classification and walking speed estimation using handheld devices". In: *Proceedings of the 2012 ACM Conference on Ubiquitous Computing - UbiComp '12* (2012), p. 113. DOI: 10.1145/2370216.2370235. URL: http://dl.acm.org/citation.cfm?doid=2370216.2370235{\%}5Cnhttp://dl.acm.org/ft{_}gateway.cfm?id=2370235{\&}ftid=1290080{\&}dwn=1{\&}CFID=216938597{\&}CFTOKEN=33552307.

[29] Jorge-L Reyes-Ortiz et al. "Transition-Aware Human Activity Recognition using smartphones." In: *Neurocomputing: An International Journal* 171 (2016), pp. 754–767. ISSN: 09252312. DOI: 10.1016/j.neucom.2015.07.085. URL: http://ovidsp.ovid.com/ovidweb.cgi?T=JS{\&}PAGE=reference{\&}D=psyc11{\&}NEWS=N{\&}AN=2015-39180-001.

[30] Daniel Roggen et al. "Collecting complex activity datasets in highly rich networked sensor environments". In: *INSS 2010 - 7th International Conference on Networked Sensing Systems* (2010), pp. 233–240. DOI: 10.1109/INSS.2010.5573462.

[31] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: (2016), pp. 1–12. arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747.

[32] Benjamin Schuster-Böckler and Alex Bateman. "An introduction to hidden Markov models". In: *Curr Protoc Bioinformatics* Appendix 3.January (1986), pp. 4–16. ISSN: 1934-340X. DOI: 10.1002/0471250953.bia03as18. URL: http://www.ncbi.nlm.nih.gov/pubmed/18428778.

[33] F Sha and F Pereira. "Shallow parsing with conditional random fields". In: *Proceedings of the 2003 Conference of the North ...* June (2003), pp. 1071–1079. DOI: 10.3115/1073445.1073473. URL: http://dl.acm.org/citation.cfm?id=1073473.

[34] Mark Stamp. "A revealing introduction to hidden Markov models". In: *Department of Computer Science San Jose State ...* (2004), pp. 1–20. DOI: 10.1.1.136.137. URL: http://scholar.google.com/scholar?hl=en{\&}btnG=Search{\&}q=intitle:A+Revealing+Introduction+to+Hidden+Markov+Models{\#}0.

[35] T Starner and A Pentland. "Real-time American Sign Language recognition from video using hidden Markov models". In: *Motion-Based Recognition* (1997), pp. 227–243. DOI: 10.1109/ISCV.1995.477012.

[36] Zuolei Sun et al. "Activity classification and dead reckoning for pedestrian navigation with wearable sensors". In: *Measurement Science and Technology* 20.1 (2009), p. 015203. ISSN: 0957-0233. DOI: 10.1088/0957-0233/20/1/015203. URL: http://stacks.iop.org/0957-0233/20/i=1/a=015203?key=crossref.0f2a56e060ab21ae68f0ec8675ce63cf.

[37] Tin Kam Ho. "Random decision forests". In: *Proceedings of 3rd International Conference on Document Analysis and Recognition* 1 (), pp. 278–282. ISSN: 0818671289. DOI: 10.1109/ICDAR.1995.598994. URL: http://ieeexplore.ieee.org/document/598994/.

[38] P. H. Veltink et al. "Detection of static and dynamic activities using uniaxial accelerometers". In: *IEEE Transactions on Rehabilitation Engineering* 4.4 (1996), pp. 375–385. ISSN: 10636528. DOI: 10.1109/86.547939. URL: http://ieeexplore.ieee.org/ielx4/86/11962/00547939.pdf?tp={\&}arnumber=547939{\&}isnumber=11962.

[39] La The Vinh et al. "Semi-Markov conditional random fields for accelerometer-based activity recognition". In: *Applied Intelligence* 35.2 (2011), pp. 226–241. ISSN: 0924669X. DOI: 10.1007/s10489-010-0216-5.

[40] Sy Bor Wang et al. "Hidden Conditional Random Fields for Gesture Recognition". In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR'06)* 2 (2006), pp. 1521–1527. ISSN: 1063-6919. DOI: 10.1109/CVPR.2006.132. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1640937.

[41] Mark Weiser. *The computer for the 21st century*. 1999. DOI: 10.1145/329124.329126. arXiv: arXiv:1011.1669v3. URL: http://portal.acm.org/citation.cfm?id=329124.329126.

[42] Wanmin Wu et al. "Classification accuracies of physical activities using smartphone motion sensors". In: *Journal of Medical Internet Research* 14.5 (2012), pp. 1–9. ISSN: 14388871. DOI: 10.2196/jmir.2208.

[43] Jhun Ying Yang, Jeen Shing Wang, and Yen Ping Chen. "Using acceleration measurements for activity recognition: An effective learning algorithm for constructing neural classifiers". In: *Pattern Recognition Letters* 29.16 (2008), pp. 2213–2220. ISSN: 01678655. DOI: 10.1016/j.patrec.2008.08.002. URL: http://dx.doi.org/10.1016/j.patrec.2008.08.002.