Wearable Sensor based Human Activity Recognition with Recurrent Neural Networks

Masterarbeit

im Masterstudiengang
Quantitative Economics

der Wirtschafts- und Sozialwissenschaftlichen Fakultät
der Christian-Albrechts-Universität zu Kiel

vorgelegt von

Markus Steger

Erstgutachter:	Prof. Dr. Matei Demetrescu
Zweitgutachter:	Prof. Dr. Kai Carstensen

Thesis advisor: Prof. Dr. Matei Demetrescu Markus Steger

Wearable Sensor based Human Activity Recognition with Recurrent Neural Networks

Abstract

This work is concerned with *human activity recognition* (*HAR*) based on signals recorded with wearable sensors. Traditional *HAR* approaches often require domain specific knowledge to generate handcrafted features, a feature selection strategy to identify the most informative ones and a supervised learning algorithm to solve the task. This thesis evaluates the practicability of *recurrent neural networks* (*RNNs*) for the *HAR* problem. *RNN* based models are well suited in theory, since they extract informative features from raw sensor data automatically, can be trained in a supervised fashion and are tailored to model temporal dynamics. The investigated architectures are based on *long short-term memory* (*LSTM*) and *gated recurrent unit* (*GRU*) networks, hybrid networks with convolutional and recurrent layers and *residual networks* (*ResNets*) applied on recurrent layers. The performance of the networks is evaluated on three public and one private dataset, covering a diverse selection of activities and subjects. All datasets were subject to a *train, validation and test split* and a *leave-one-subject-out-cross-validation* (*LOSOCV*) evaluation procedure. In order to have a valid comparison with the traditional *HAR* approach, a *random forest* (*RF*) classifier was trained on time and frequency domain features. Furthermore, a grid search was carried out for all models, based on a validation set, to find the best hyperparameters. Results show that the proposed networks outperform the traditional approach in both evaluation procedures on three out of four datasets and are competitive in the remaining one. This suggests that the time extensive process of generating handcrafted features based on domain specific knowledge can be avoided by relying on *RNNs*. However, any *neural network* (*NN*) based model requires careful hyperparameter optimization, which in turn is a very time consuming process with an uncertain prospect of success.

Contents

Listing of Abbreviations

Abbreviation	Description
ADL	Activity of Daily Living
CNN	Convolutional Neural Network
CONV_LSTM	Convolutional Neural Network with LSTM Layers on Top
CRF	Conditional Random Field
DFT	Discrete Fourer Transformation
ECG	Electrocardiography
FN	False Negative
FP	False Positive
GPU	Graphical Processing Unit
GRU	Gated Recurrent Unit
Hadamard Product	Entry-wise Product
HAR	Human Activity Recognition
IID	Independent and Identically Distributed
IQR	Interquartile Range
LOSOCV	Leave One Subject Out Cross Validation
LSTM	Long Short-Term Memory
MMA	Mixed Martial Arts
NAG	Nesterov's Accelerated Gradient
NESW	North East South West
NN	Neural Network
PDF	Probability Density Function
ReLu	Rectifier Linear Unit
RES_LSTM	Residual Network based on LSTM Layers
ResNet	Residual Network
RF	Random Forest
RNN	Recurrent Neural Network
SSE	Sum of Squared Errors
SVM	Support Vector Machine
TN	True Negative
TP	True Positive

Listing of figures

Listing of tables

1

Introduction

1.1 Motivation

Human activity recognition (*HAR*) is a key research area in ubiquitous computing with a diverse area of applications, such as sports analytics, entertainment and health care. Therefore, *HAR* has not only the potential to have a significant impact on society, but also presents a challenging area of research due to the complex and diverse nature of human activities (Lara & Labrador, 2013).

Wearable sensors are placed on several body parts and typically consist of an acceleration device, a gyroscope device, and sometimes a magnetometer as well. A traditional

approach to *HAR* tasks includes feature engineering based on segments of the recorded time series data, a feature selection procedure, and a supervised classification algorithm (Bulling et al., 2014). However, generating informative features can be a challenging and time consuming task, which often requires domain specific knowledge. Additionally, the information content of the handcrafted features depends on the task at hand and could vary across different activities and datasets. This can be considered as the major drawback of the traditional approach to *HAR*.

Neural networks (*NN*s) became one of the biggest trends in machine learning research over the last decade. This kind of model can automatically extract informative features from raw sensor recordings and therefore has the potential to replace the traditional *HAR* approach. *Recurrent neural networks* (*RNN*s) in particular might be suited well to tackle *HAR*, since this subclass of *NN*s are designed to model temporal dynamics in the data. Recent advances in the methodology of *NN*s and specifically *RNN*s led to various state-of-the-art performance results in different domains, such as image recognition (Krizhevsky et al., 2012), speech recognition (Graves et al., 2013) and natural language processing (Zhang & Zong, 2015).

The motivation of the present work is to replace the traditional approach to *HAR* with an automated end-to-end procedure in the form of *RNN*s, which does not require any specific domain knowledge. Furthermore, this thesis tries to leverage *RNN* based models to achieve at least a competitive performance as compared to the traditional *HAR* approach.

1.2 Objective

This thesis identifies the practicability of *RNN*s for sensor based *HAR*. Several *RNN* architectures are evaluated on three public datasets and one private dataset. The models either consist of recurrent layers, a combination of convolution and recurrent layers, or residual layers. The latter is a concept introduced by He et al. (2016) that was originally applied on convolution layers. For the scope of the present work, the concept of residuals will be applied on recurrent layers and is, to the best of my knowledge, a novel application in the context of *HAR*.

Additionally, the performance of the *RNN*s will be compared to the traditional *HAR* approach, which is based on typical time and frequency domain features, in combination with a *random forest* (*RF*) classification algorithm.

1.3 Structure

The remaining work of this thesis is structured as follows:

Chapter 2 - Sensor Based Activity Recognition: This chapter presents background information on *HAR*, describes the physical measurements of wearable sensors and introduces a few application use cases.

Chapter 3 - Theoretical Background: This part of the thesis formally introduces neural networks and further formalizes the relevant architectures.

Chapter 4 - Related Work: In this chapter a literature review is conducted and the recent advances in *HAR* are outlined.

Chapter 5 - Experiments and Results: This chapter describes the conducted experiments and discusses the results.

Chapter 6 - Conclusion and Future Work: The present work is concluded and potential future research directions are suggested.

2

Sensor Based Activity Recognition

2.1 Human Activity Recognition

HAR aims to identify actions or activities of single or multiple users in a real life environment. There are two main approaches to *HAR* that can be distinguished by the sensor type. The first one involves sensors placed in the environment, such as video cameras and audio microphones. These sensor placements lead to interesting applications in computer vision and speech recognition. The second approach uses one or more on-body sensors to recognize the activity (Bulling et al., 2014) and is the focus of this thesis.

Over the last 20 years, advances in the semiconductor industry led to wearable sen-

sors that are small in size and more robust to material failure, while their battery lifetime and memory capacity steadily increases (Bonato, 2003). Apart from the fact that on-body sensors are rather low priced as compared to video or audio systems, their major advantage lies in the area of privacy. Sensor data cannot be interpreted as easily as video or sound streams and therefore are more compatible with a world where privacy is becoming increasingly important.

2.2 Physical Measurements

On-body sensors typically consist of an acceleration device, a gyroscope device and sometimes a magnetic field device.

Acceleration devices measure the change of speed of some mass in three orthogonal axes x, y and z. The measurement unit is either meters per square second $\frac{m}{s^2}$ or g-force $g\frac{m}{s^2}$, where $g \approx 9.81$ is the gravity on earth's surface (Kok et al., 2017).

Gyroscope devices are used to measure angular velocity in three orthogonal axes. Angular velocity is defined as the rate of change of the angular position of a rotating object with respect to time. It is either measured in radians per second or degrees per second (Kok et al., 2017).

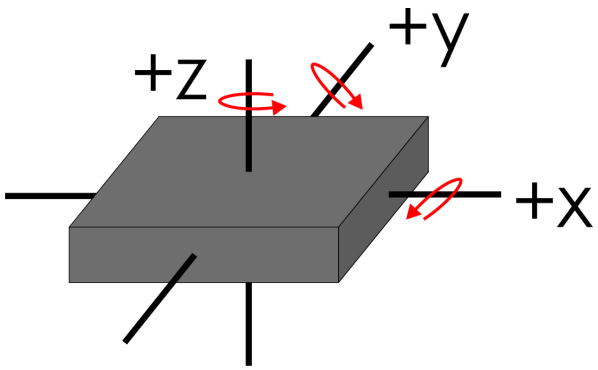


Figure 2.1: Acceleration and Gyroscope Device. Figure taken from Autonomous Robots Lab (2018).

Figure 2.1 visualizes both concepts, where the red circles denote the angular velocity in a specific axis. Hence, given both measurements are sufficiently accurate and an initial starting position is known, the position of an object can be tracked over time (Kok et al., 2017).

Finally, a magnetometer can be understood as a digital compass that measures the direction and the magnitude of the magnetic field relative to the device. Therefore, the device can be used to estimate the absolute position of the device in the *NESW* plane. The measurement units are typically in gauss or microtesla (Coillot & Leroy, 2012).

2.3 Applications

2.3.1 Health Care and Assisted Living

The range of potential applications of *HAR* is extensive and penetrates many different aspects of life. However, the areas of application with the most significant impact on society are arguably in health care and assisted living.

Jiang et al. (2008) propose CareNet, which is a wireless sensor networking environment for remote health care. This system is tracking physiological signals, such as blood oxygen levels and heart activity (*ECG*) and other critical information including movement and fall detection. The core idea behind this project is to shift health care from a traditional clinical setting to the patient's home environment.

Another critical application concerns dementia patients. Osmani et al. (2007) propose a system based on environmental and on-body sensors, which uses *HAR* and a reminding system to assist patients with various degrees of dementia. A similar approach was used by Kautz et al. (2002) to help patients with Alzheimer's disease.

A last example involves stroke patients. Bartalesi et al. (2005) propose a system using kinesthetic wearable sensors in combination with *HAR* to detect gestures executed by the upper limbs.

2.3.2 Sports and Entertainment

Wearable sensors were used in recent years in the sports and entertainment section for recognition of sportive and leisure activities to improve lifestyle quality and entertainment experience.

There are currently several commercial systems on the market for performance tracking and monitoring of sportive activities. For example, Nike Inc. provides a sensor for running shoes, which logs several running exercises and keeps track of the training his-

tory. Polar Electro presents a similar product, but for a greater variety of activities, such as running, cycling and team sports, where the focus is on physiological signals and position tracking. However, most of the newer devices start to leverage acceleration and gyroscope data to estimate the intensity of the exercises and the user's efficiency (Lockhart et al., 2012).

Another popular application of sensor based *HAR* is in the field of context aware gaming. *HAR* is used to recognize gestures and activities of the users and therefore introduces a new way for them to interact with the game. Prime examples are the Nintendo Wii (Schlömer et al., 2008) and augmented reality systems in general (Papagiannakis et al., 2008).

3

Theoretical Background

3.1 Statistical Learning Theory

The current chapter assumes the reader has a basic knowledge in statistical learning theory. Therefore, this section will only provide a short summary of key concepts in statistical learning. In case the reader needs a more rigorous refresher on the topic, James et al. (2014) and Hastie et al. (2001) are recommended to read.

The objective of statistical learning is to construct a data-driven predictive model, which generalizes well on unseen data and, if possible, gives insights into the underlying relationship structures within the data. Hence, statistical learning is about *prediction* and

inference procedures (Hastie et al., 2001).

There are several different learning categories, such as *reinforcement learning*, *unsupervised learning* and *supervised learning*. The present work concentrates only on the last category.
Since supervised learning can be further distinguished into *regression* and *classification* problems, the *HAR* problem in this thesis is framed as a time series classification task. A variety of linear and nonlinear models are capable to tackle this kind of a problem. However, the former type of model leads to a *generalization error* decomposition with a high *bias* and a low *variance*, while the latter achieves a low *bias* and a high *variance* due to its nonlinear nature. This is a central problem in supervised learning and is known as the *bias-variance tradeoff*. Therefore, the model choice is an empirical issue and must be determined based on the performance of the considered models on the given dataset (James et al., 2014).

In order to ensure that a model is not *overfitting* on the training data and therefore generalizes well on unseen data, the out-of-sample performance has to be measured. A typical evaluation approach involves either *cross-validation* or splitting the data into *train*, *validation and test sets* (Hastie et al., 2001). This work makes use of both approaches and describes each evaluation procedure in section 5.3.2.

3.2 Neural Networks

This section presents a class of estimators known as neural networks. This type of model can be applied in the context of statistical learning to solve problems framed in a super-

vised fashion.

NNs try to approximate a function $\mathbf{T} = f(\mathbf{X})$, which maps the input matrix $\mathbf{X}$ to the target vector $\mathbf{T}$. The network introduces the mapping $\mathbf{T} = f(\mathbf{X}; \boldsymbol{\theta})$ with $\boldsymbol{\theta}$ being a vector containing all the parameters of the network. These parameters will then be adjusted until a sufficient mapping is achieved.

NNs consist of several interconnected processing units, which can be organized into layers. The connections are typically directed and therefore can be visualized as a directed graph. Each unit uses an affine transformation of its inputs, applies a nonlinear transformation and pushes the resulting output through all outgoing connections to the next layer of units (Goodfellow et al., 2016).

A single network layer can be formalized in general terms as follows,

$$G(\mathbf{X}) = \sum_i V_i \sigma(\mathbf{W}_i^T \mathbf{X}) \tag{3.1}$$

where $\mathbf{W}_i, \mathbf{X} \in \mathbb{R}^D$, $V_i \in \mathbb{R}$, $\mathbf{W}^T\mathbf{X}$ is the inner product and σ is a nonlinear transformation applied element-wise. Cybenko (1989) proofed that this type of function belongs to the class of universal approximators and stated the following theorem:

Theorem 1 *Let σ be any continuous sigmoidal function, denote I_D as the D-dimensional unit hypercube $[0,1]^D$ and let $C(I_D)$ refer to the space of continuous functions on I_D. Then finite sums of the form $G(\mathbf{X})$ are dense in $C(I_D)$. This means, given any $f \in C(I_D)$ and $\varepsilon > 0$, there is a sum $G(\mathbf{X})$ satisfying*

$$|G(\mathbf{X}) - f(\mathbf{X})| < \varepsilon \qquad \forall \quad \mathbf{X} \in I_n \tag{3.2}$$

This theorem was generalized by Sun & Cheney (1992) and Light (1992) to hold for σ that are nonconstant, bounded and monotonically-increasing continuous functions. Thus, NNs can learn any desired input-output mapping if the appropriate parameters are found. However, it is important to mention that many methods share this property, such as kernel based models, tree based models and splines (Wassermann, 2006). Further note that finding the appropriate parameters is a non-trivial process; especially in the case of NNs, which can contain several million parameters.

3.2.1 Feedforward Networks

The structure of a fully connected feedforward network can be seen in the following figure:

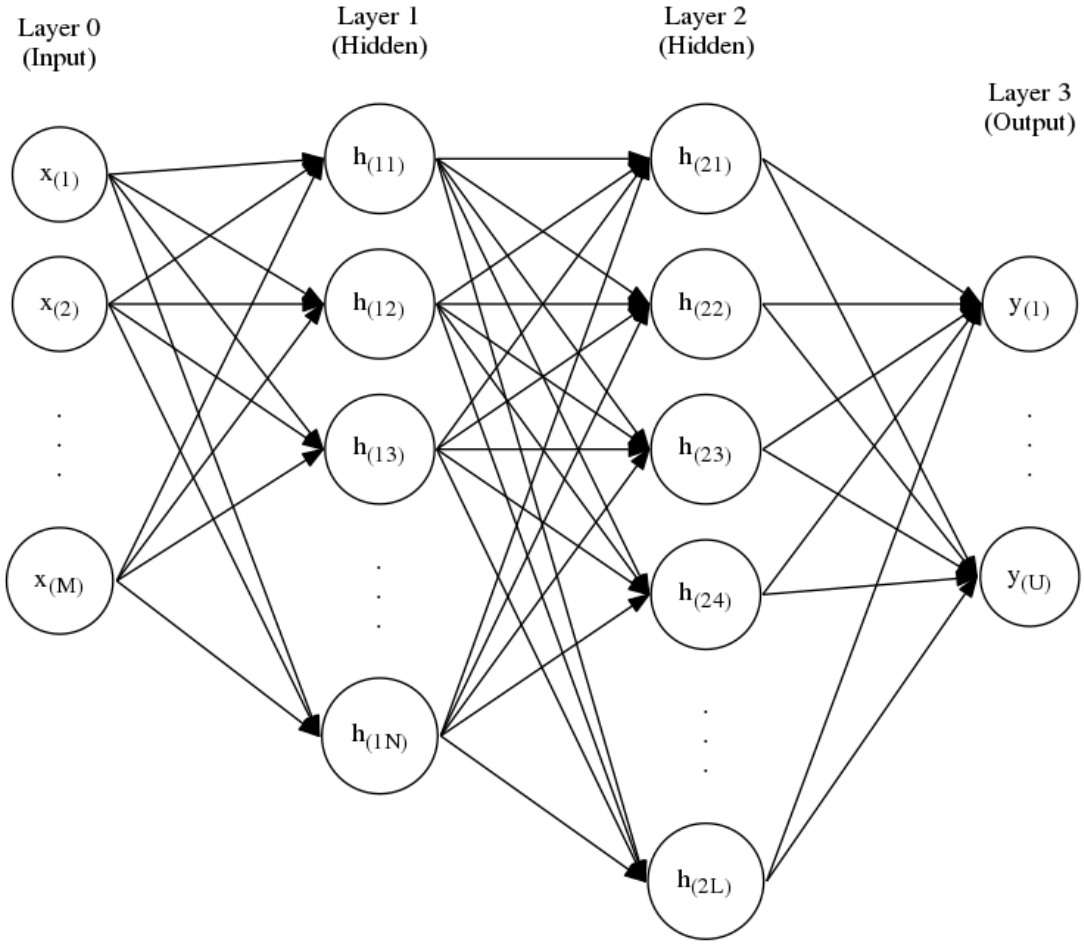


Figure 3.1: Two-Layer Fully Connected Feedforward Neural Network. Figure is based on Bishop (2006).

The network displayed in figure 3.1 starts with an input layer on the left, two hidden layers in the middle and an output layer on the right. In principle, the number of layers, number of units per layer and the density of the connections are hyperparameters that

can be completely arbitrary (Bishop, 2006).

Each unit takes a weighted combination of all its input connections, which are usually from the previous layer, and applies a nonlinear transformation known as the activation function. Thus, the network is pushing the original input from layer to layer in a feedforward manner (Goodfellow et al., 2016).

A fully connected feedforward network can be formalized as follows: The ordering of the layers and of their corresponding units is given by,

$$\mathcal{L} = \{l \in \mathbb{Z}_0^+ \mid l \leq L\} \tag{3.3}$$

$$\mathcal{U}_l = \{u_l \in \mathbb{Z}^+ \mid u_l \leq U_l\} \tag{3.4}$$

where L is the last layer of the network and U_l is the number of units in layer l. Hence, the i-th unit in layer l is defined as,

$$h_i^{(l)} = \sigma_{(l)}\Big(\sum_{j \in \mathcal{U}_{l-1}} w_{ij}^{(l)} h_j^{(l-1)} + b_i^{(l)}\Big) \tag{3.5}$$

where $w_{ij}^{(l)}$ is the weight associated with the input connection from unit $h_j^{(l-1)}$ of the previous layer to the current unit and $\sigma_{(l)}$ is the activation function used by all units in layer l. The term $b_i^{(l)}$ is called bias and fulfills the same purpose as the intercept in linear regression. Further note that the first hidden layer $h^{(1)}$ takes incoming connections directly from the input layer denoted as x. Hence, $h^{(0)} \equiv x$.

The network output, the last layer in the network, is computed in a similar fashion:

$$y_i \equiv h_i^{(L)} = \sigma_{(L)}\Big(\sum_{j \in \mathcal{U}_{L-1}} w_{ij}^{(L)} h_j^{(L-1)} + b_i^{(L)}\Big) \tag{3.6}$$

In principle, the activation function σ in the hidden and output layers can be the same, but usually differs in practice (Bishop, 2006). The formal description of the network can be further simplified if matrix notation is used as in Goodfellow et al. (2016):

$$\mathbf{H}^{(l)} = \sigma_{(l)}\big(\mathbf{W}^{(l)T}\mathbf{H}^{(l-1)} + \mathbf{B}^{(l)}\big) \qquad \forall l \in \mathcal{L} \setminus \{0, L\} \tag{3.7}$$

$$\mathbf{Y} \equiv \mathbf{H}^{(L)} = \sigma_{(L)}\big(\mathbf{W}^{(L)T}\mathbf{H}^{(L-1)} + \mathbf{B}^{(L)}\big) \tag{3.8}$$

where the weight matrix of layer l is defined as $\mathbf{W}^{(l)} \in \mathbb{R}^{(U_l \times U_{l-1})}$ with U_l hidden units and U_{l-1} incoming connections from the previous layer. The bias term is formalized as $\mathbf{B}^{(l)} \in \mathbb{R}^{(U_l \times 1)}$, but must be broadcasted into $\mathbf{B}^{(l)} \in \mathbb{R}^{(U_l \times J)}$ to match the dimensions. Since the activation function σ is applied element-wise, the hidden and output layers can be defined as $\mathbf{H}^{(l)}, \mathbf{Y}^{(L)} \in \mathbb{R}^{(U_l \times J)}$. Further note that in case of the first hidden layer $\mathbf{H}^{(1)}$, all incoming connections are from the input layer $\mathbf{X}$, which is defined as $\mathbf{X} \in \mathbb{R}^{(M \times J)}$ with M features and J samples.

3.2.2 Network Training

This section will introduce an optimization procedure for neural networks that will try to find appropriate weights such that a sufficient input-output mapping is achieved. The procedure of finding weights is often referred to as model training in the machine learning community (Ripley, 2007).

3.2.2.1 Gradient Descent Optimization

Before presenting the actual network training algorithm, the underlying optimization technique will be introduced.

Gradient descent is an iterative first-order optimization procedure that tries to minimize a given objective function by following the negative gradient. An example is given by the following objective function and its partial derivative:

$$y = f(x) = 3.2 + 1.2 * (x - 2)^2 \tag{3.9}$$

$$\frac{dy}{dx} = 2.4 * (x - 2) \tag{3.10}$$

The optimization procedure approximates the minimum of y by iterating the following equation,

$$x_i = x_{i-1} - \lambda \frac{dy}{dx}\Big|_{x=x_{i-1}} \tag{3.11}$$

where λ is the learning rate or the so-called step size. x_i and x_{i-1} are the current and the previous solution respectively and $\frac{dy}{dx}$ is the first-order derivative of y with respect to x, evaluated at the previous solution. *Gradient descent* is starting at some initial solution x_0 and is updating its solution by following the negative gradient. This procedure will be repeated until a sufficient solution is obtained (Bishop, 2006). The algorithm is visualized in the following figure,

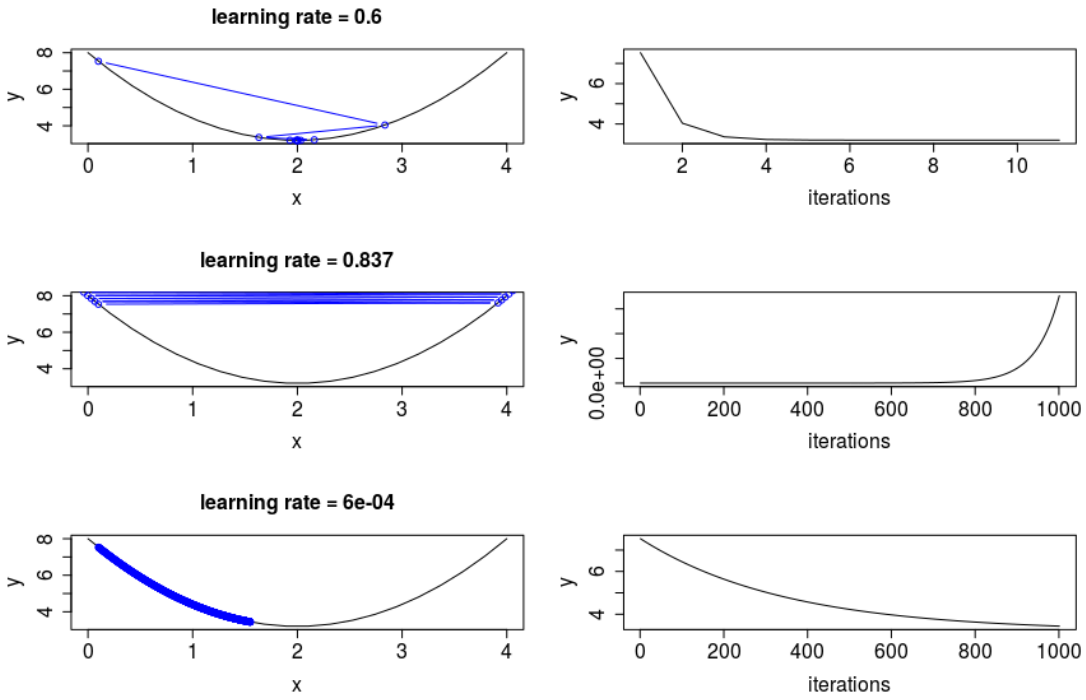


Figure 3.2: Left: Gradient Descent Trajectories with Different Learning Rates. Right: Number of Iterations until Convergence.

where it is apparent that a solution close to the analytical one was found after only a few iterations (learning rate = 0.6). However, if the learning rate is too big, the algorithm can get stuck oscillating in a valley and will not converge to a sufficient solution. Otherwise, if the learning rate is too small, the speed of convergence is very slow and a sufficient solution might not be found in time.

In general, the rate of convergence will become slow in the neighborhood of any stationary point, because the gradient will be very small. Further issues arise if the objective function has multiple stationary points. In that case, *gradient descent* can overshoot local minima and even converge to local maxima (Reed & Marks, 1999).

3.2.2.2 Forward and Backward Propagation

The training method for *NN*s is called *backpropagation* or simply *backprop* and is based on *gradient descent*. The algorithm involves a forward pass, which computes the network output with a given set of weights, and a backward pass that adjusts the weights by computing the gradient of the objective function with respect to all weights.

The starting point of the *backprop* algorithm is some cost or objective function,

$$C(t, y) = \sum_{i \in U_L} \sum_{b \in \mathbb{B}} C_{ib}(y_{ib}, t_{ib}) \tag{3.12}$$

which computes some error based on the target t and the network prediction y for all output units and all training examples in the batch $\mathbb{B} = \{b \in \mathbb{Z}^+ \mid b \leq |\mathbf{X}|\}$. Hence, the batch size can vary from a single sample to all available training data. The former is known as *online learning*, while the latter is called *batch learning*. Furthermore, a single iteration or epoch of the *backprop* algorithm is completed when every sample in the training data was processed once (Goodfellow et al., 2016).

First note that the sample index b will be dropped in favor of a simpler notation. Now, we assume a fully connected neural network with L layers and σ_l as the corresponding activation function for all units in layer l. The forward pass for output unit i equals equation 3.6, but can be rewritten as follows,

$$y_i = \sigma_L\Big(\sum_{j=1}^{U_{L-1}} w_{ij}^{(L)} h_j^{(L-1)} + b_i^{(L)}\Big) = \sigma_L\Big(\sum_{j=0}^{U_{L-1}} w_{ij}^{(L)} h_j^{(L-1)}\Big) = \sigma_L\big(z_i^{(L)}\big) \tag{3.13}$$

where the bias term is absorbed by the sum in such a way that there is an additional weight $w_{i0}^{(L)}$ with the corresponding $h_0^{(L-1)} = 1$ term. Additionally, the weighted combination of all incoming connections is now defined as $z_i^{(L)}$. Further note, that the above equation is recursive and can be expanded until the input layer is reached (Bishop, 2006).

In order to derive the backward pass, the partial derivatives with respect to all weights must be computed. For example, the partial derivative with respect to $w_{ij}^{(L)}$, which connects the hidden unit j of layer $L-1$ with the output unit i, can be computed as follows:

$$\frac{\partial C}{\partial w_{ij}^{(L)}} = \frac{\partial C}{\partial y_i} \frac{\partial y_i}{\partial z_i^{(L)}} \frac{\partial z_i^{(L)}}{\partial w_{ij}^{(L)}} = C'(y_i)\sigma_L'(z_i^{(L)})h_j^{(L-1)} \tag{3.14}$$

Please note that C' and σ_L' are the first order derivatives w.r.t. y_i and $z_i^{(L)}$ respectively. Further, the error signal from the objective function will be denoted as

$$\delta_i^{(L)} = C'(y_i)\sigma_L'(z_i^{(L)}) \tag{3.15}$$

Hence, equation 3.14 simplifies to:

$$\frac{\partial C}{\partial w_{ij}^{(L)}} = \delta_i^{(L)} h_j^{(L-1)} \tag{3.16}$$

Taking the derivative with respect to $w_{jk}^{(L-1)}$ is slightly more complex. Since this weight is connecting a unit in layer $L-2$ with a unit in layer $L-1$, the error signal must be

propagated backwards through all affected network paths.

$$\begin{aligned}\frac{\partial C}{\partial w_{jk}^{(L-1)}} &= \sum_{i=1}^{U_L} \frac{\partial C}{\partial y_i} \frac{\partial y_i}{\partial z_i^{(L)}} \frac{\partial z_i^{(L)}}{\partial h_j^{(L-1)}} \frac{\partial h_j^{(L-1)}}{\partial z_j^{(L-1)}} \frac{\partial z_j^{(L-1)}}{\partial w_{jk}^{(L-1)}} \\ &= \sum_{i=1}^{U_L} C'(y_i, t_i) \sigma_L'(z_i^{(L)}) w_{ij}^{(L)} \sigma_{L-1}'(z_j^{(L-1)}) h_k^{(L-2)} \\ &= \sigma_{L-1}'(z_j^{(L-1)}) \sum_{i=1}^{U_L} \delta_i^{(L)} w_{ij}^{(L)} h_k^{(L-2)} \end{aligned} \tag{3.17}$$

The error signal corresponding to some specific unit in layer $l < L$ can be generalized as follows,

$$\delta_j^{(l)} = \sigma_l'(z_j^{(l)}) \sum_{i=1}^{U_{l+1}} \delta_i^{(l+1)} w_{ij}^{(l+1)} \tag{3.18}$$

where the sum considers all units i of layer $l + 1$, to which unit j of layer l is connected. The corresponding generalization of the evaluated derivative can then be denoted as (Bishop, 2006):

$$\frac{\partial C}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} h_j^{(l-1)} \tag{3.19}$$

Hence, equation 3.17 simplifies to,

$$\frac{\partial C}{\partial w_{jk}^{(L-1)}} = \delta_j^{(L-1)} h_k^{(L-2)} \tag{3.20}$$

Finally, the weight update equation presented in section 3.2.2.1 is applied to every weight

in the network:

$$w_{ij}^{(l)} = w_{ij}^{(l)} - \lambda \frac{\partial C}{\partial w_{ij}^{(l)}} \tag{3.21}$$

The *backprop* algorithm can be summarized as follows:

1. Propagate the input through the network by computing equation 3.6 for all output units.
2. Compute the error signal δ_i for all output units using equation 3.15.
3. Backpropagate the error signal to get δ_j for all remaining units in the network using equation 3.18.
4. Evaluate the derivatives using equation 3.19.
5. Update all weights using equation 3.21.
6. Repeat until some stopping criteria is reached.

3.2.2.3 Vanishing and Exploding Gradients

This section addresses a well known issue with the *backprop* algorithm that can slow down the training process and even stop the whole training procedure altogether.

We consider again a fully connected feedforward network with L layers. The *back-*

propagation formula for the error signal can be analyzed as follows:

$$\delta_i^{(l)} = \sigma_l'(z_i^{(l)}) \sum_{k=1}^{U_{l+1}} \delta_k^{(l+1)} w_{ki}^{(l+1)} \tag{3.22}$$

$$\begin{aligned} \delta_j^{(l-1)} &= \sigma_{l-1}'(z_j^{(l-1)}) \sum_{i=1}^{U_l} \delta_i^{(l)} w_{ij}^{(l)} \\ &= \sigma_{l-1}'(z_j^{(l-1)}) \sum_{i=1}^{U_l} \sigma_l'(z_i^{(l)}) w_{ij}^{(l)} \sum_{k=1}^{U_{l+1}} \delta_k^{(l+1)} w_{ki}^{(l+1)} \end{aligned} \tag{3.23}$$

Equation 3.23 can be further expanded until the output layer is reached. It is important to note that the number of terms of $\sigma'(z)$ and w in equation 3.23 will grow proportionally to the number of layers that are needed to reach the output layer. Please also note that w is determined by the network itself and therefore is not a hyperparameter unlike $\sigma'(z)$. Thus, we can analyze the corresponding limit behavior of $\sigma'(z)$,

$$\lim_{p \to \infty} \left(\sigma'(z)\right)^p = \begin{cases} \infty & \text{for } \sigma'(z) > 1 \\ 1 & \text{for } \sigma'(z) = 1 \\ 0 & \text{for } \sigma'(z) < 1 \end{cases} \tag{3.24}$$

where $\sigma'(z)$ is assumed to be a constant. It is apparent that the choice of the activation function and specifically its first order derivative are crucial for the weight updates in lower layers. The propagated error signal might become extremely small or big, depending on how many layers are needed to reach the output layer and what kind of activation function is used in the respective layers. A more rigorous treatment of this issue can be found in Hochreiter (1998).

3.2.3 Activation Functions

This section introduces common functional forms for the activation function of the output and hidden units and discusses their properties.

3.2.3.1 Output Activations

Typical functional forms for the output activation depend on the task at hand. For regression problems, the identity function is often selected, because the output does not have to be bounded. In contrast, classification tasks require the output to be a vector of class probabilities. Thus, a sigmoid function is often used for binary classification and a softmax function for multi-class classification (Hastie et al., 2001).

At first, the weighted combination term of a single unit will be recalled:

$$z_i = \sum_{j=1}^{U_{l-1}} w_{ij}^{(l)} h_j^{(l-1)} + b_i^{(l)} \tag{3.25}$$

The activation functions for a output unit can be formalized as follows:

$$\sigma_{sigm}(z_i) = \frac{1}{1 + e^{(-z_i)}} \tag{3.26}$$

$$\sigma_{soft}(z_i) = \frac{e^{(z_i)}}{\sum_i e^{(z_i)}} \tag{3.27}$$

Both activation functions allow a proper probabilistic interpretation of the output. The output range is $(0, 1)$, while the input range is $(-\infty, \infty)$. Please note that the remaining class probability in a binary case can be computed with $1 - \sigma_{sigm}(z_i)$ and that equation 3.27 enforces a network output vector that sums to unity (Bishop, 2006).

3.2.3.2 Hidden Activations

*NN*s were originally inspired by the brain and therefore the sigmoid function was also used as activation for the hidden units, since it resembled the behavior of firing neurons (Patterson, 1996). Please note that the first derivative of the sigmoid function is

$$\frac{d}{dz_i}\sigma_{sigm}(z_i) = \sigma_{sigm}(z_i)(1 - \sigma_{sigm}(z_i)) \tag{3.28}$$

and the corresponding visualization can be seen in the following figure:

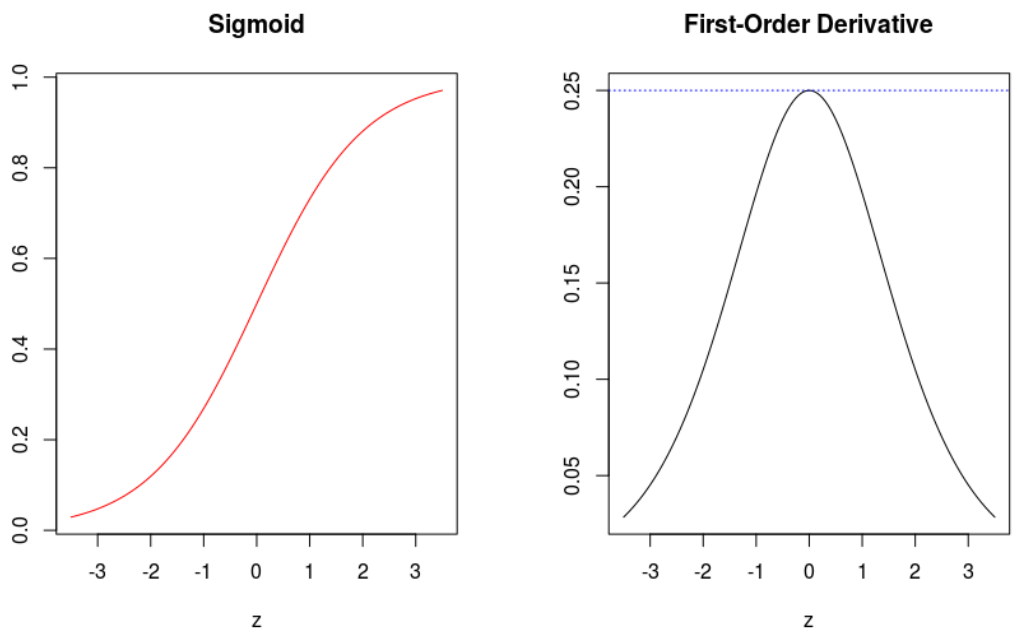


Figure 3.3: Left: Sigmoid Function. Right: First Derivative of Sigmoid Function.

The main characteristics of the sigmoid function can be described by the non-negative

output, the linear region around zero with the maximum first-order derivative of 0.25 and the saturation regions at zero and one (Marsland, 2009). In this context, saturation means that the first-order derivative evaluated at some input is very small. These characteristics affect the training algorithm in a negative fashion and often lead to the vanishing gradient problem as outlined in 3.2.2.3.

Another common hidden activation is the tanh function, which is defined and visualized as follows:

$$\sigma_{tanh}(z_i) = \frac{cosh(z_i)}{sinh(z_i)} = \frac{e^{z_i} - e^{-z_i}}{e^{z_i} + e^{-z_i}} = 2\sigma_{sigm}(2z_i) - 1 \tag{3.29}$$

$$\frac{d}{dz_i}\sigma_{tanh}(z_i) = 1 - \sigma_{tanh}(z_i)^2 \tag{3.30}$$

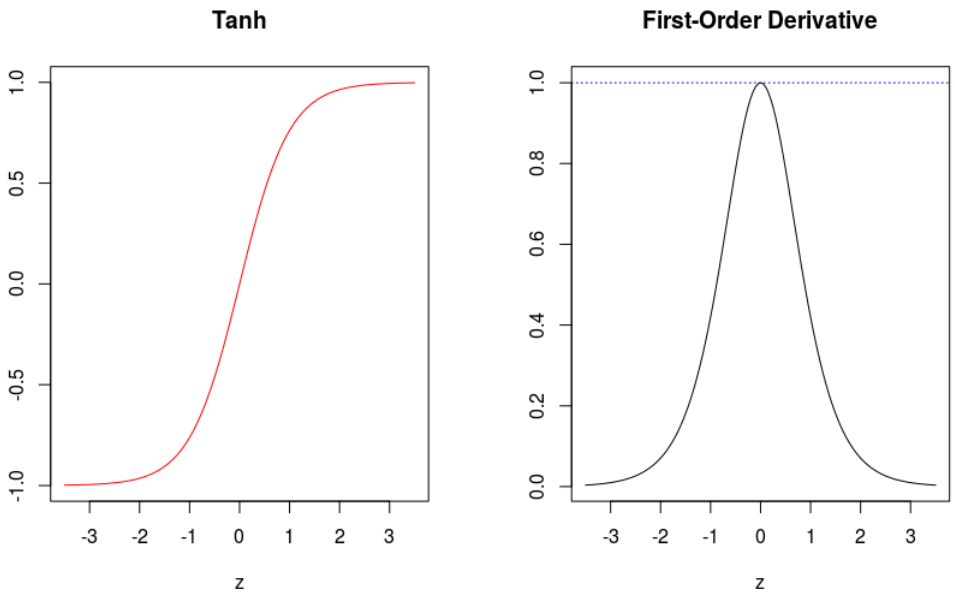


Figure 3.4: Left: Tanh Function. Right: First Derivative of Tanh Function.

Equation 3.29 shows that the tanh function is a rescaled version of the sigmoid function that exhibits rotational symmetry around zero. This function maps from the input range $(-\infty, \infty)$ to the output range $(-1, 1)$, has a linear region around zero with a maximum derivative of one and saturates at ± 1. These characteristics affect the training algorithm less severe than in the case of the sigmoid activation. However, the vanishing gradient problem is still predominant (Goodfellow et al., 2016).

A more recent activation function for hidden units is called rectifier or *rectifier linear unit* (*ReLu*). This type of activation has several variations. The most basic one is defined and visualized as follows:

$$\sigma_{rect}(z_i) = z_i^+ = max(0, z_i) = \begin{cases} z_i & \text{for } z_i \geq 0 \\ 0 & \text{for } z_i < 0 \end{cases} \tag{3.31}$$

$$\frac{d}{dz_i}\sigma_{rect}(z_i) = \begin{cases} 1 & \text{for } z_i \geq 0 \\ 0 & \text{for } z_i < 0 \end{cases} \tag{3.32}$$

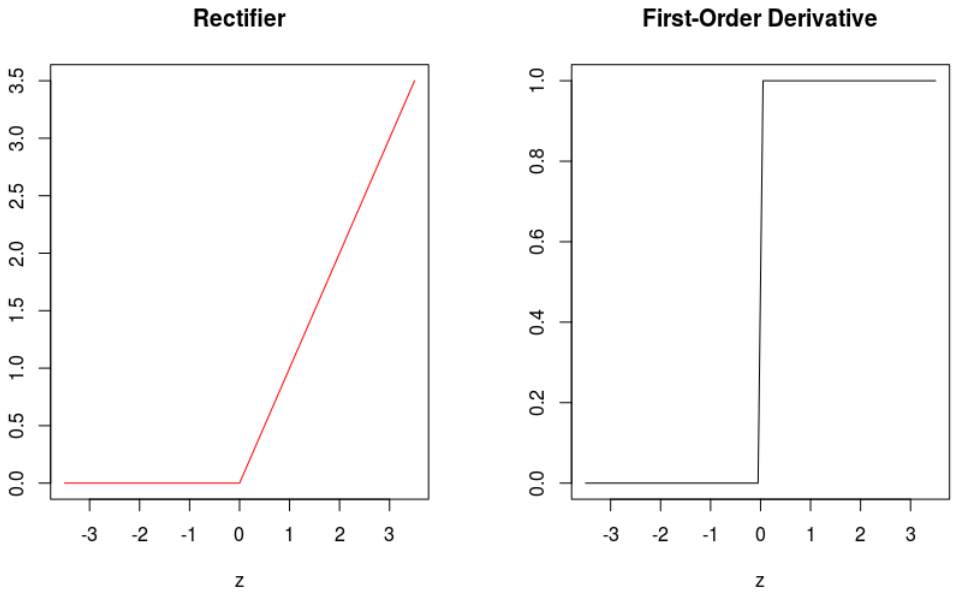


Figure 3.5: Left: Rectifier Function. Right: First Derivative of Rectifier Function.

The rectifier activation maps from the input range $(-\infty, \infty)$ to the output range $[0, \infty)$ and has a constant derivative of one for non-negative inputs. This activation function is not affected by the vanishing gradient problem and therefore has a big advantage over any sigmoid based activation. Additionally, *ReLu* allows a network to obtain a sparse structure, which leads to the fact that some units output a zero and are completely shutdown. This characteristic has the potential to speed up the training algorithm and to make the network robust to small input changes as well. The latter can lead to significant performance gains. Furthermore, this activation function is computationally more efficient since there are no expensive operations involved such as the exponential function (Goodfellow et al., 2016).

However, the rectifier function is not differentiable at zero, which might impact the

training algorithm. Therefore, several variations were proposed like the softplus function, the leaky rectifier and the noisy rectifier (Glorot et al., 2011), (Maas et al., 2013), (He et al., 2015). But since these variations only lead to marginal performance improvements in practice, they will not be covered in the scope of this thesis.

3.2.4 Weight Initializations

*NN*s are often trained by first-order optimization algorithms such as *gradient descent* and therefore require the initialization of the weights.

In order to avoid saturation of the units in the early stages of the training process, a proper initialization strategy is essential. Simply initializing the network with equal weights cannot be recommended as all units in a layer will compute the same output and thus receive the same weight update. Hence, it is important to randomly initialize the weights, otherwise the whole layer will behave as a single unit (LeCun et al., 1998).

A common approach is to derive a parametric distribution based on the activation function to ensure that most weights are outside the saturation range of the chosen activation function (Reed & Marks, 1999). An example derivation can be found in the appendix A.1.

In practice, the *Glorot initialization* is an often used strategy. This initialization regime is derived in a similar fashion, but is motivated by the observation that large networks are quickly effected by the vanishing gradient problem during the early stages of the training

process due to the multiplication effect[1]. The Glorot initialization tries to solve this issue with the following consideration: During the forward propagation part, the input and output variances of the units should be equal. Thus, the units in deeper layers are not pushed into the saturating regions of the activation function, nor do they get amplified upon successive passes through the layers. A similar logic applies to the backward pass. The variances of the gradients should be equal across layers, so that they do not explode or vanish too early during the training process. This consideration leads to the following parametric distribution, which should be used in combination with the tanh activation,

$$w_i \sim \mathcal{U}\left(\frac{-\sqrt{6}}{\sqrt{N_{in}+N_{out}}}, \frac{\sqrt{6}}{\sqrt{N_{in}+N_{out}}}\right) \tag{3.33}$$

where N_{in} and N_{out} are the number of incoming connections and the number of outgoing connections of some unit. A detailed derivation of this initialization regime can be found in Glorot & Bengio (2010).

A more recent method is called *He initialization*. This initialization strategy builds upon the previously introduced Glorot method, but is tailored towards the rectifier activation. He et al. (2015) proposed the following parametric distribution for weight initialization:

$$w_i \sim \mathcal{N}\left(0, \sqrt{\frac{2}{N_{in}}}\right) \tag{3.34}$$

The detailed derivation can be found in their paper.

[1] See section 3.2.2.3.

3.2.5 Objective Functions

In order to train a neural network with *backpropagation*, a cost or objective function must be specified as a measure of fit. It must be noted that any cost function has to be differentiable, so that the *backprop* algorithm is applicable.

A very common choice for regression problems is the *sum of squared errors* (*SSE*), which is defined as

$$C(t, y) = \sum_{i \in U_L} \sum_{b \in \mathbb{B}} C_{ib}(t_{ib}, y_{ib}) = \sum_{i \in U_L} \sum_{b \in \mathbb{B}} (y_{ib} - t_{ib})^2 \tag{3.35}$$

where t_{ib} represents the target and y_{ib} denotes the network prediction (Hastie et al., 2001). In a classification problem, the *cross-entropy* loss function is typically deployed,

$$C(t, y) = \sum_{i \in U_L} \sum_{b \in \mathbb{B}} C_{ib}(t_{ib}, y_{ib}) = - \sum_{i \in U_L} \sum_{b \in \mathbb{B}} t_{ib} \ln \left(y_{ib} \right) \tag{3.36}$$

where y_{ib} can be understood as being the probability of some sample b belonging to the class i. Note that the ground truth t_{ib} is binary. Hence, cross-entropy is measuring the dissimilarity between two distributions (Bishop, 2006).

3.2.6 Enhanced Optimization Methods

This section introduces some extensions of the *gradient descent* method with the objective of improving training speed and model performance. The presented methods will be split into procedures, which update the learning rate globally, and procedures, which adjust the learning rate for each weight separately.

3.2.6.1 Global Learning Rate Updates

The weight update equation described in section 3.2.2.2 will be slightly redefined with a less complex notation,

$$w^{(\tau+1)} = w^{(\tau)} - \lambda \frac{\partial C(w^{(\tau)})}{\partial w^{(\tau)}} \tag{3.37}$$

where τ is the number of the iteration, w is some weight within the network and λ is the learning rate. Please note that λ is a fixed constant and is independent of the iteration. However, using a fixed learning rate for all parameters at every iteration can cause a slow convergence or even no convergence at all, as described in 3.2.2.1.

Momentum tackles this problem by incorporating information about recent gradient changes. A so-called velocity term is added to the previous equation, which accelerates the *gradient descent* procedure in the direction of persistent reductions in the objective function. The momentum method is given by:

$$w^{(\tau+1)} = w^{(\tau)} - \lambda \frac{\partial C(w^{(\tau)})}{\partial w^{(\tau)}} + \alpha \overbrace{\left(w^{(\tau)} - w^{(\tau-1)}\right)}^{velocity} \tag{3.38}$$

$$\Delta w^{(\tau+1)} = -\lambda \frac{\partial C(w^{(\tau)})}{\partial w^{(\tau)}} + \alpha \Delta w^{(\tau)} \tag{3.39}$$

where $\alpha \in [0, 1]$ is the momentum coefficient. The velocity term can be seen as an exponential average of all previous gradients that is smoothing out the weight updates. Thus, if successive weight changes point in the same gradient direction, the velocity terms reinforce each other and therefore accelerate the *gradient descent* procedure. Otherwise, if the gradient changes direction continuously, the velocity term will level out the variation

(Sutskever et al., 2013). In order to see this effect, the velocity term can be expanded as follows,

$$\begin{aligned}\Delta w^{(\tau+1)} &= -\lambda \frac{\partial C(w^{(\tau)})}{\partial w^{(\tau)}} + \alpha\big(-\lambda \frac{\partial C(w^{(\tau)})}{\partial w^{(\tau-1)}} + \alpha \Delta w^{(\tau-1)}\big) \\ &= -\lambda \frac{\partial C(w^{(\tau)})}{\partial w^{(\tau)}} + \alpha\big(-\lambda \frac{\partial C(w^{(\tau)})}{\partial w^{(\tau-1)}} + \alpha\big(-\lambda \frac{\partial C(w^{(\tau)})}{\partial w^{(\tau-2)}} + \cdots\big)\big) \\ &= -\lambda \sum_{k=0}^{\infty} \alpha^k \frac{\partial C(w^{(\tau)})}{\partial w^{(\tau-k)}}\end{aligned} \tag{3.40}$$

where α controls how long recent terms influence the average (Reed & Marks, 1999).

Nesterov's accelerated gradient (*NAG*) is closely related to the previously introduced momentum extension. While momentum computes the gradient with respect to the current position $w^{(\tau)}$ in weight space, *NAG* evaluates the gradient at the approximate future position. Thus, the velocity term can respond to changes in a quicker fashion. The weight update equation with *NAG* is given by (Sutskever et al., 2013):

$$\Delta w^{(\tau+1)} = -\lambda \frac{\partial C(w^{(\tau)} + \alpha w^{(\tau)})}{\partial w^{(\tau)}} + \alpha \Delta w^{(\tau)} \tag{3.41}$$

3.2.6.2 Adaptive Learning Rate Methods

The following category of procedures tries to adjust the learning rate λ dynamically for each parameter at each iteration. Therefore, extensive manual optimization of the learning rate can be avoided. However, all of those procedures introduce new hyperparameters, which in turn require tuning.

For simplicity, we state $\frac{\partial C(w^{(\tau)})}{\partial w^{(\tau)}} \equiv g_{(\tau)}$ and rewrite the weight update equation 3.37

as:

$$w^{(\tau+1)} = w^{(\tau)} - \lambda g_{(\tau)} \tag{3.42}$$

AdaGrad is an adaptive procedure, which scales the learning rate λ by the L^2 norm of the historical gradients (Duchi et al., 2011). The resulting update equation is defined as:

$$w^{(\tau+1)} = w^{(\tau)} - \frac{\lambda}{\sqrt{\sum_{i=1}^{\tau} g_{(i)}^2}} g_{(\tau)} \tag{3.43}$$

Since the sum in the denominator will increase with every iteration, the learning rate will gradually become smaller. This can be seen as a major disadvantage, since λ can get too small in later iterations.

RMSProp is an unpublished procedure proposed by Geoffrey Hinton and addresses the drawback of AdaGrad by replacing the sum of gradients with an exponentially decaying average (Tieleman & Hinton, 2012). This average is constructed as follows,

$$m_{(\tau)} = p m_{(\tau-1)} + (1-p) g_{(\tau)}^2 \tag{3.44}$$

where $m_{(\tau)}$ can be interpreted as an estimate of the second-order uncentered moment of g_τ and p is the decay parameter. Hence, the weight update becomes

$$w^{(\tau+1)} = w^{(\tau)} - \frac{\lambda}{\sqrt{m_{(\tau)} + \varepsilon}} g_{(\tau)} \tag{3.45}$$

where ε is a small constant for numerical stability[2].

AdaDelta was proposed by Zeiler (2012) and further builds upon RMSProp by replacing the learning rate with a ratio of recent weight and gradient updates. The resulting update method is given as,

$$v^{(\tau)} = pv^{(\tau-1)} + (1-p)(\Delta w^{(\tau)})^2 \tag{3.46}$$

$$w^{(\tau+1)} = w^{(\tau)} - \frac{\sqrt{v^{(\tau-1)} + \varepsilon}}{\sqrt{m_{(\tau)} + \varepsilon}} g_{(\tau)} \tag{3.47}$$

where $v_{(\tau)}$ is an estimate of the second-order uncentered moment of $\Delta w^{(\tau)}$. Please note that this method is insensitive to unexpected gradient spikes, since the numerator lags behind the denominator. Therefore, the denominator can react faster and reduce the learning rate for the current iteration.

Adam was proposed by Kingma & Ba (2014) and is closely related to the idea behind RMSProp. However, Adam uses estimates of first and second order moments, which both decay over time. The moment estimates of the gradient and the resulting weight

[2] ε is intended to enforce progress in case previous weight updates have become very small.

update equation are given as,

$$a_{(\tau)} = \beta_1 a_{(\tau-1)} + (1-\beta_1) g_{(\tau)} \tag{3.48}$$

$$q_{(\tau)} = \beta_2 q_{(\tau-1)} + (1-\beta_2) g^2_{(\tau)} \tag{3.49}$$

$$\hat{a}_{(\tau)} = \frac{a_{(\tau)}}{1-\beta_1^{(\tau)}} \tag{3.50}$$

$$\hat{q}_{(\tau)} = \frac{q_{(\tau)}}{1-\beta_2^{(\tau)}} \tag{3.51}$$

$$w^{(\tau+1)} = w^{(\tau)} - \lambda \frac{\hat{a}_{(\tau)}}{\sqrt{\hat{q}_{(\tau)}} + \varepsilon} \tag{3.52}$$

where $\hat{a}_{(\tau)}$ and $\hat{q}_{(\tau)}$ are bias correction terms for the moment estimates, since these values are initialized with zeros and hence biased towards zero. The βs are the decay parameters and ε is a small value for numerical stability.

3.2.7 Regularization Techniques

This section presents techniques that help neural networks to avoid overfitting and therefore improve their generalization capabilities on unseen data.

Regularization techniques in general work by trading low bias for reduced variance. A typical approach is to add a penalty term to the objective function, such that the model capacity is limited in some form. Thus, the regularized objective function can be defined in generic terms as,

$$\mathbb{C}(\mathbf{X}, \mathbf{T}; \boldsymbol{\theta}) = \mathbf{C}(\mathbf{X}, \mathbf{T}; \boldsymbol{\theta}) + \psi \boldsymbol{\Omega}(\boldsymbol{\theta}) \tag{3.53}$$

where $\mathbf{X}$ and $\mathbf{T}$ are the input and target vectors, $\boldsymbol{\Omega}(\boldsymbol{\theta})$ is the regularization term, $\psi \in$

$[0, \infty)$ controls the impact of the regularization and $\boldsymbol{\theta}$ is the parameter vector (Hastie et al., 2001).

L_2 Regularization, also known as *weight decay*, is a common parameter norm penalty. The rational behind this concept is to keep the weights as small as possible, so the network does not react too sensitive to small changes in the input. The regularization term is given by:

$$\boldsymbol{\Omega}(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{\theta}\|_2^2 \tag{3.54}$$

Hence, the cost function and the weight update equation are given as,

$$\mathbb{C}(X, T; \theta) = \mathbf{C}(\mathbf{X}, \mathbf{T}; \boldsymbol{\theta}) + \psi\frac{1}{2}\|\boldsymbol{\theta}\|_2^2 \tag{3.55}$$

$$\begin{aligned}\boldsymbol{\theta} &= \boldsymbol{\theta} - \lambda\big(\psi\boldsymbol{\theta} + \nabla_\theta \mathbf{C}(\mathbf{X}, \mathbf{T}; \boldsymbol{\theta})\big) \\ &= (1 - \lambda\psi)\boldsymbol{\theta} - \lambda\nabla_\theta \mathbf{C}(\mathbf{X}, \mathbf{T}; \boldsymbol{\theta})\big)\end{aligned} \tag{3.56}$$

where the weight decay term shrinks the weight vector in each iteration by a constant factor. Thus, weights which do not decrease the cost function significantly will be pushed towards zero (Goodfellow et al., 2016).

L_1 Regularization is very similar to the previous norm penalty and is defined as:

$$\boldsymbol{\Omega}(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{\theta}\|_1 \tag{3.57}$$

The main difference between L_1 and L_2 *regularization* is that the former can lead to a

sparse network, where the weights can actually become zero. In contrary, the L_2 regularization can only lead to weights, which are close to zero. A more detailed analysis of the properties can be found in (Bishop, 2006).

Early Stopping is an alternative approach to regularization. This technique is constantly observing the model performance on some validation set during training and stops the training process as soon as some stopping criteria is reached. The weights of the best iteration are then retrieved and used for the prediction on the test set. However, the validation set must represent the test set reasonable well and the stopping criteria should allow some sort of patience. The latter is necessary since neural networks do not improve monotonically with each iteration. There can be several iterations without improvement until another performance increase is achieved. The technique can be summarized as follows:

1. Split data into train, validation and test sets.
2. Train the neural network on the train set.
3. Compute the performance on the validation set every i-th iteration.
4. Continue model training until the performance metric is not improving for *j* iterations or the maximum number of iterations is reached.
5. Retrieve the weights of the best iteration and compute the performance on the test set.

Please note that a good performance on the validation set does not necessarily imply a good performance on the test set. Hence, the performance measure on the validation set is no reliable measure of the generalization capabilities of the model (Hastie et al., 2001).

Dropout takes a different approach on regularization compared to any previously introduced method. The procedure combines the predictions of several different networks at test time. This is done by randomly ignoring some units during training. Hence, the dropout technique can be seen as a model averaging method, which is a common way of tackling the problem of overfitting (Bishop, 2006).

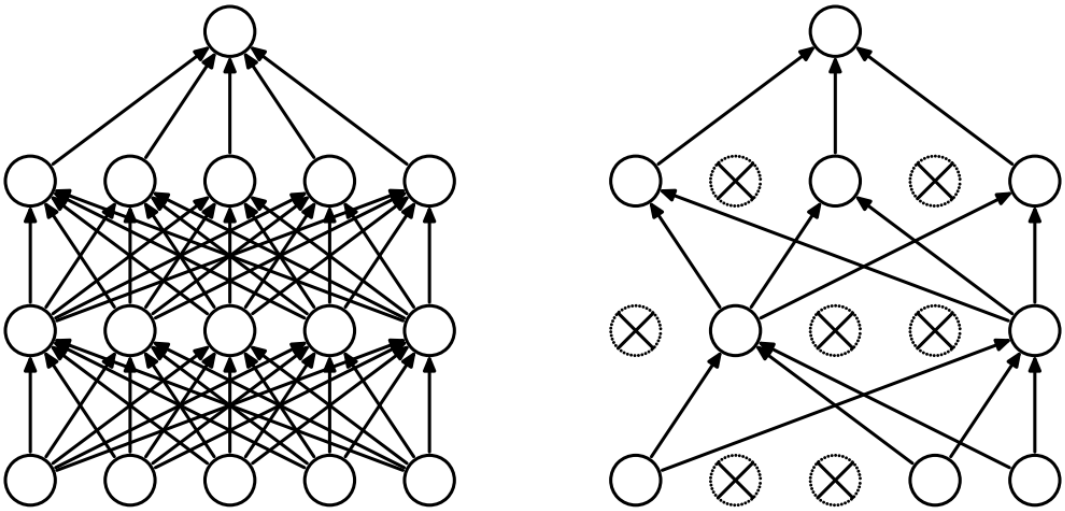


Figure 3.6: Left: Fully Connected Feedforward Network. Right: After Dropout was applied. Figure taken from Srivastava et al. (2014).

More specifically, units are dropped randomly with probability p at each training iteration, but at test time all units are active. Thus, the outgoing weights for each unit must be rescaled to make sure that the expected output of a unit is equal to the actual output at test time. A very basic approach is to simply multiply the outgoing weights by p^{-1} (Srivastava et al., 2014).

3.3 Neural Network Extensions

3.3.1 Recurrent Networks

RNNs are designed to model temporal dynamics in time series data by introducing loops in their architecture (Goodfellow et al., 2016).

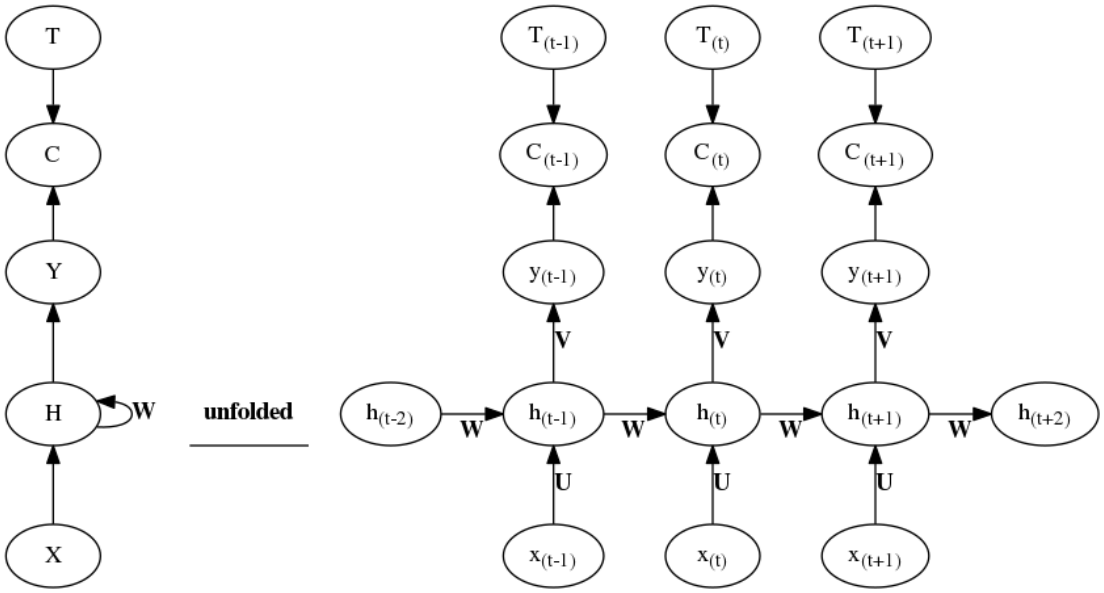


Figure 3.7: Left: RNN with a single hidden Layer. Right: RNN unfolded in time. Figure is based on Goodfellow et al. (2016).

The above figure visualizes a typical *RNN* architecture, where **X** is the input matrix, **H** is the hidden layer, **Y** is the network output, **T** is the target vector and C is the cost function of network output and target. Further note that $\mathbb{T} = \{t \in \mathbb{Z}^+ \mid t \leq T\}$ defines the time index and shall not be confused with the target vector **T**. **U** is the weight matrix that represents input-to-hidden unit connections, **W** is the weight matrix that denotes hidden-to-hidden recurrent connections and **V** is the weight matrix that contains hidden-to-output connections. It is important to note that **U**, **V** and **W** are shared between the

timesteps to keep the number of parameters as small as possible. This is known as *parameter sharing* and enables the network to be trained with less data (Bishop, 2006).

Figure 3.7 shows a *RNN* with one hidden layer that produces an output for each timestep. However, in some use cases like time series classification, it might be sufficient to only produce a single output. Therefore the hidden layer can be modified to only push the last timestep to the output layer. Since h_T is a function of all previous timesteps, it should be the most informed one (in theory) and therefore might contain sufficient information to successfully solve the time series classification task.

A *RNN* with a single hidden layer can be formalized for $\forall t \in \mathbb{T}$ as follows,

$$\mathbf{h}_t = \sigma_1(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{h}_{t-1} + \mathbf{b}_1) \tag{3.58}$$

$$\mathbf{y}_t = \sigma_2(\mathbf{V}\mathbf{h}_t + \mathbf{b}_2) \tag{3.59}$$

where $\mathbf{h}_t$ is the hidden layer at timestep t, σ is applied element-wise and $\mathbf{b}$ is the bias vector for the respective layer. Further note that $\mathbf{h}_0$ is usually initialized with zeros. The above equations show a single hidden layer to keep the notation simple, but this structure can be generalized to an arbitrary number of layers (Goodfellow et al., 2016).

An interesting insight can be generated by visualizing a *RNN* as a feedforward network,

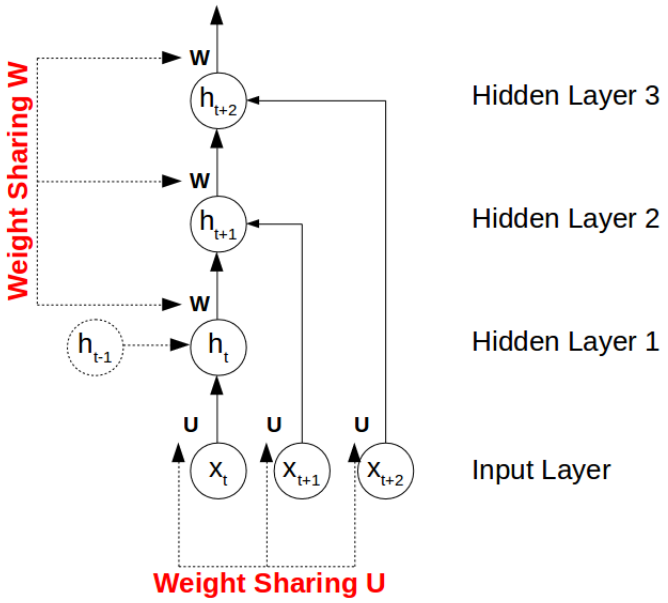


Figure 3.8: RNN visualized as a Feedforward Network.

where it is easy to see that each layer represents one timestep. Hence, the number of layers are directly proportional to the sequence length and therefore are prone to vanishing and exploding gradients for long sequences. Furthermore, this representation shows that the *backprop* algorithm is applicable, although the gradient must be additionally propagated through time.

3.3.1.1 Long Short-Term Memory Networks

The *LSTM* network was originally proposed by Hochreiter & Schmidhuber (1997) and tackles the vulnerability of *RNN*s regarding the vanishing and exploding gradients by replacing the recurrent units with *LSTM* cells.

Several gates inside a *LSTM* cell control how the input is processed and stored in an internal memory state. A visualization of a *LSTM* cell can be seen in the following figure,

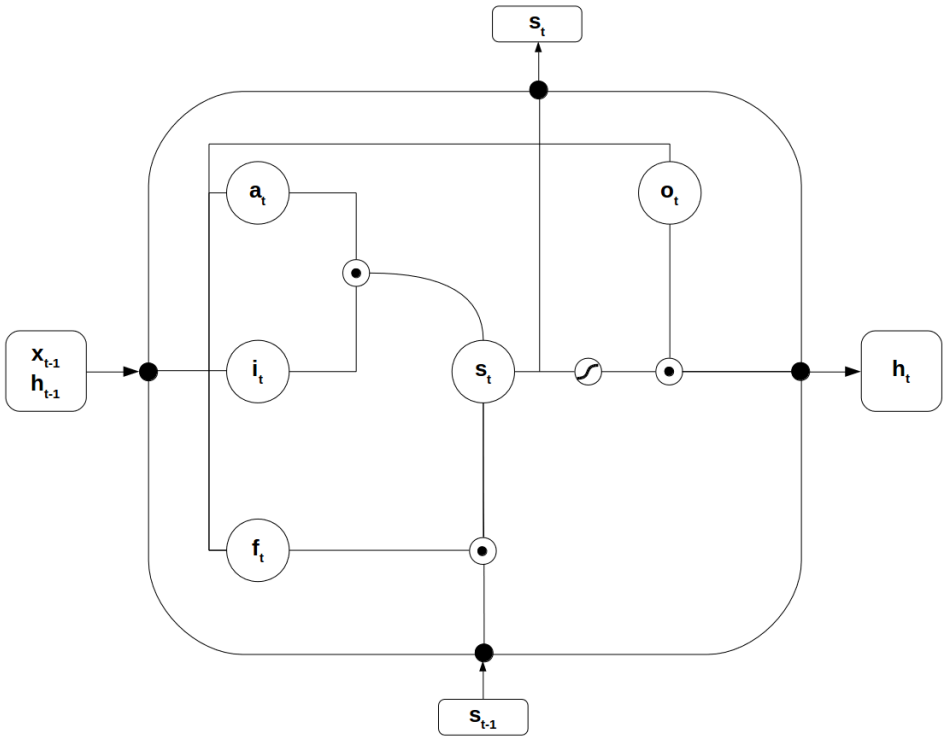


Figure 3.9: LSTM Cell. Figure is based on Hochreiter & Schmidhuber (1997).

where $\mathbf{a}_t$ is the input activation, $\mathbf{i}_t$ the input gate, $\mathbf{f}_t$ the forget gate and $\mathbf{o}_t$ the output gate. Additionally, an internal memory is denoted as $\mathbf{s}_t$ and the cell output as $\mathbf{h}_t$. The internal state $\mathbf{s}_t$ can be understood as an inner self-loop additional to the outer self-loop in regular *RNNs*. However, the inner self-loop is controlled by several gates. While the forget gate allows the cell to forget previous memory states, the input gate regulates how much the current state shall be altered by the incoming signal. The final cell output $\mathbf{h}_t$ can then further be altered by the output gate (Hochreiter & Schmidhuber, 1997),

(Goodfellow et al., 2016).

The *LSTM* cell can be formalized with the following equations at time t,

$$\mathbf{a}_t = \sigma_{tanh}\left(\mathbf{W}_a\mathbf{x}_t + \mathbf{U}_a\mathbf{h}_{t-1} + \mathbf{b}_a\right) \tag{3.60}$$

$$\mathbf{i}_t = \sigma_{sigm}\left(\mathbf{W}_i\mathbf{x}_t + \mathbf{U}_i\mathbf{h}_{t-1} + \mathbf{b}_i\right) \tag{3.61}$$

$$\mathbf{f}_t = \sigma_{sigm}\left(\mathbf{W}_f\mathbf{x}_t + \mathbf{U}_f\mathbf{h}_{t-1} + \mathbf{b}_f\right) \tag{3.62}$$

$$\mathbf{o}_t = \sigma_{sigm}\left(\mathbf{W}_o\mathbf{x}_t + \mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{b}_o\right) \tag{3.63}$$

and the corresponding state and cell output definitions,

$$\mathbf{s}_t = \mathbf{a}_t \odot \mathbf{i}_t + \mathbf{f}_t \odot \mathbf{s}_{t-1} \tag{3.64}$$

$$\mathbf{h}_t = \sigma_{tanh}(\mathbf{s}_t) \odot \mathbf{o}_t \tag{3.65}$$

where $\odot$ is the *Hadamard product*. The corresponding *backprop* algorithm is derived in A.2.

In order to understand why the *LSTM* network is hardly affected by vanishing or exploding gradients, we recall a simplified version (one dimension) of the *RNN* recurrency in equation 3.58 and its partial derivative with respect to some timestep $t' > t$ in the past:

$$h_t = \sigma(wh_{t-1}) \tag{3.66}$$

$$\frac{\partial h_{t'}}{\partial h_t} = \prod_{k=1}^{t'-t} w\sigma'(wh_{t'-k}) \tag{3.67}$$

It is apparent that the whole partial derivative can become very small or very big, de-

pending on the behavior of w and σ'. If the weight term is not unity, it can either shrink or grow exponentially fast. A similar influence has the choice of activation function σ, which was already described in section 3.2.2.3. Now, we compare the previous equation to the inner recurrency of the *LSTM* cell state, equation 3.64,

$$\frac{\partial s_{t'}}{\partial s_t} = \prod_{k=1}^{t'-t} f_{t+k} = \prod_{k=1}^{t'-t} \sigma_{sigm}(z_{t+k}) \tag{3.68}$$

where z_{t+k} is the input of the forget gate. The above equation lacks the exponential factor associated with the weights and the recurrent link s_{t-1} is activated by the identity function. Hence, there is no intrinsic factor pushing the derivative to zero. Furthermore, since the forget gate maps its input into the output range $[0, 1]$, it cannot take values greater than one. This further helps the *LSTM* to contain the problem of exploding gradients (Bayer, 2015), (Pascanu et al., 2013).

3.3.1.2 Gated Recurrent Units

*GRU*s were proposed by Cho et al. (2014) and can be understood as a simplified variation of *LSTM* networks.

The *GRU* combines the input and forget gate and merges the internal memory with

the cell output. The resulting model is defined as follows,

$$\mathbf{u}_t = \sigma_{sigm}\left(\mathbf{W}_u\mathbf{x}_t + \mathbf{U}_u\mathbf{h}_{t-1} + \mathbf{b}_u\right) \tag{3.69}$$

$$\mathbf{r}_t = \sigma_{sigm}\left(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1} + \mathbf{b}_r\right) \tag{3.70}$$

$$\mathbf{k}_t = \sigma_{tanh}\left(\mathbf{W}_k\mathbf{x}_t + \mathbf{U}_k(\mathbf{r}_t \odot \mathbf{h}_{t-1})\right) \tag{3.71}$$

$$\mathbf{h}_t = (1 - \mathbf{u}_t) \odot \mathbf{h}_{t-1} + \mathbf{u}_t \odot \mathbf{k}_t \tag{3.72}$$

where $\mathbf{u}_t$ is the update gate and $\mathbf{r}_t$ is the reset gate. The latter determines in the definition of $\mathbf{k}_t$ how the new signal should be combined with the previous memory and the update gate defines in $\mathbf{h}_t$ how much of the previous memory should be kept as cell output. The main differences compared to the *LSTM* are:

- *GRU* has two gates, whereas the *LSTM* has three gates.
- Input and forget gates are merged into an update gate.
- *GRU* has no explicit internal memory. The memory is merged with the cell output.

There are a lot more different variations of gated networks. However, Jozefowicz et al. (2015) found that no other variant could outperform neither the *LSTM* nor the *GRU* across different datasets in a consistent fashion.

3.3.2 Convolutional Networks

Convolutional neural networks (*CNNs*) were proposed by LeCun et al. (1989) and are designed to exploit grid-like topology in the data. Typical use cases involve time series data, which can be seen as a 1*D* grid with samples at certain time intervals, and image data that can be thought of as a 2*D* grid of pixels (width, height). Another example is video data,

which can be viewed as a 3*D* grid (width, height, time) (Goodfellow et al., 2016).

We start by introducing the convolution operation in 1*D* as follows,

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da \tag{3.73}$$

where $x(t)$ is some measurement w.r.t. time t, $w(a)$ is a weighting function and a is the age of a specific measurement. The objective is to smooth out $x(t)$, because it is suspected to be a noisy measurement. If $w(a)$ is a proper *probability density function* (*PDF*), then $s(t)$ can be seen as a weighted average that gives more weight to recent measurements. Thus, we assume that very distant measurements are unrelated to the current measurement, which is reasonable in a lot of use cases e.g. position tracking of a moving object with sensor data.

Since measurements are finite and not continuous in real applications, the convolution operation can be stated in discrete form:

$$s(t) = (x * w)(t) = \sum_{a=0}^{T} x(a)w(t-a) \tag{3.74}$$

Furthermore, the convolution can be applied to multiple dimensions,

$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m, j-n) \tag{3.75}$$

where I is a 2*D* input and K is a 2*D* kernel. The latter is just another terminology for the weighting function $w(a)$. Please note that convolutions are commutative and therefore

can be stated as follows,

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n) \tag{3.76}$$

where the kernel is flipped relative to the input.

*CNN*s replace regular units with convolutions, where the kernel entries are treated as network weights. The kernel is behaving like a sliding window that is shifting through the input in several dimensions. Each window processed by the kernel can be seen as a linear combination, which in turn will be put through a nonlinear activation like the rectifier. The final output of a convolution layer is then called feature map. A 2*D* convolution example is visualized in the following figure:

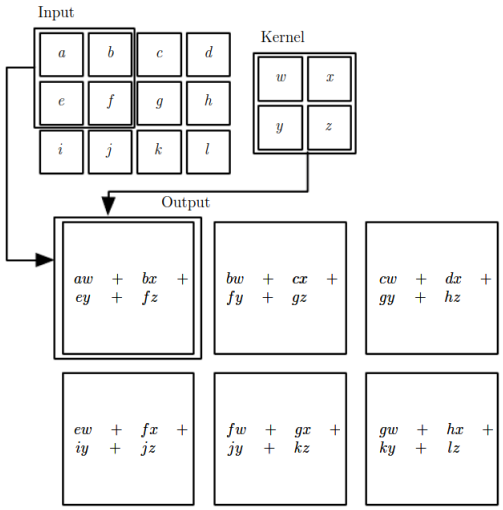


Figure 3.10: 2*D* Convolution with a single Stride. The Convolution Operation is casting the Input from $\mathbb{R}^{(3x3)}$ to $\mathbb{R}^{(2x3)}$. Figure is taken from Goodfellow et al. (2016).

Compared to traditional feedforward networks, *CNN*s leverage three key concepts: *sparse connectivity*, *parameter sharing* and *translation invariance*. *Sparse connectivity* refers

to the fact that the kernel is usually smaller than the input. Hence, the kernel processes only a specific local region of the input in each sliding step and therefore has a small number of connections (weights). And since the *CNN* will use the same kernel entries for all regions in the input matrix, the network applies *parameter sharing*. This further reduces the total number of weights in the network, which enables the model to be trained with less data. Finally, *translation invariance* occurs when a pooling layer is put on top of a convolution layer. The pooling layer uses summary statistics of local regions in the feature map, so that the feature map becomes robust to minor changes in the input. Additionally, the number of weights, which connect the feature map with the next layer, can be further reduced if a sufficiently big pooling size in combination with a sufficiently big stride length is used (Goodfellow et al., 2016). A common choice for the pooling operation is called max pooling (Zhou & Chellappa, 1988) and is visualized in the following figure:

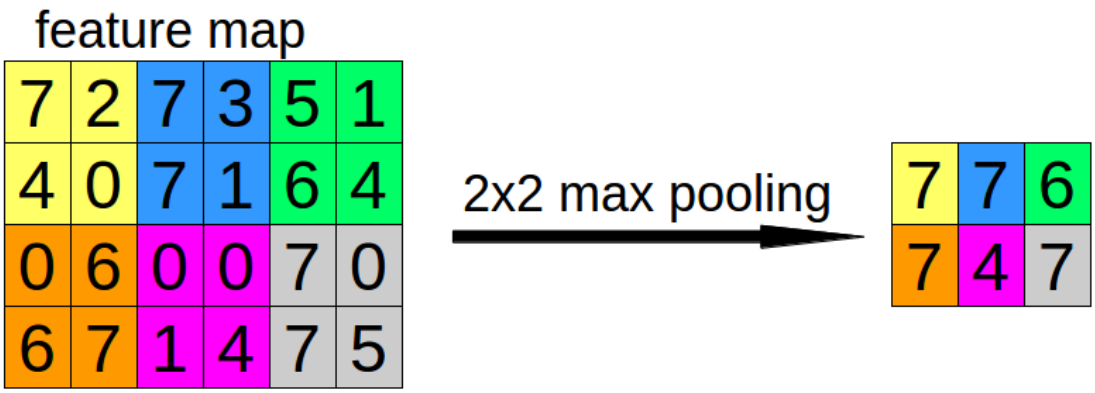


Figure 3.11: 2x2 Max Pooling applied to a Feature Map of size 4x6 with Stride s=2. Figure is based on LeCun et al. (1989).

In order to conclude this section, a typical structure of an *CNN* will be displayed,

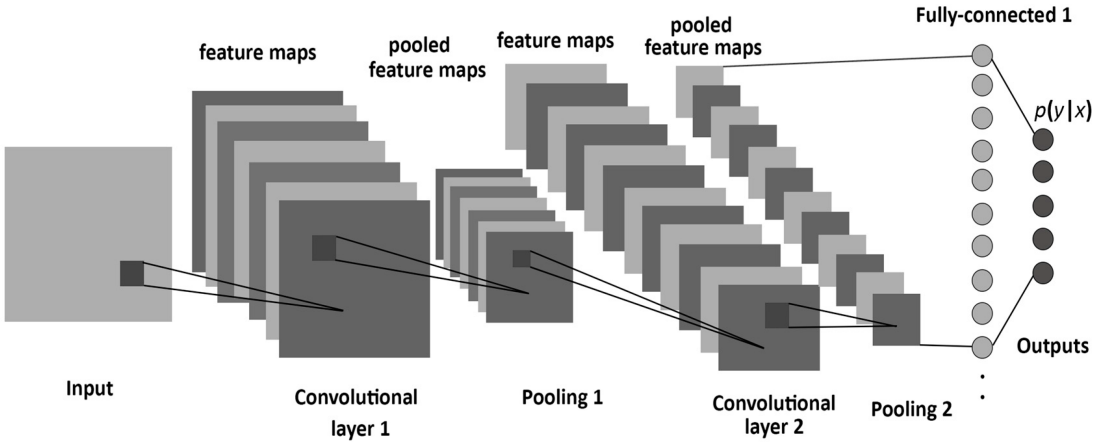


Figure 3.12: Convolutional Neural Network with Pooling Layers and a fully Connected Output Layer. Figure is taken from Albelwi & Mahmood (2017).

with two convolutional layers, two pooling layers and a fully connected output layer.

3.3.3 Residual Networks

*ResNet*s were recently proposed by He et al. (2016) and can be seen as a breakthrough in the computer vision community. This type of network enables the practitioner to train networks with several hundred layers in an efficient fashion.

The core idea behind *ResNet*s is focused on networks with many layers. If a network has more than the sufficient amount of layers to efficiently solve some task, the redundant layers should not degrade the performance of the network. Instead of further transforming the data, the redundant layers should just pass the data to the output layer. He et al. (2016) hypothesized that it might be easier for neural networks to push the weights of some redundant layer to zero than to reconstruct a linear mapping with

nonlinear functions. Hence, they introduced identity connections, which skip one or more hidden layers. Formally, this can be described as follows,

$$\mathbf{H} = F(\mathbf{X}) \tag{3.77}$$

where $\mathbf{X}$ is some input matrix, possibly the output of some previous layer and $F(\mathbf{X})$ is the nonlinear transformation of the hidden layer. Now, introducing the skip connection alters the previous equation to:

$$\mathbf{H} = F(\mathbf{X}) + \mathbf{X} \tag{3.78}$$

where the input $\mathbf{X}$ is simply added to the hidden layer output. The visualization of a skip connection can be seen in the following figure,

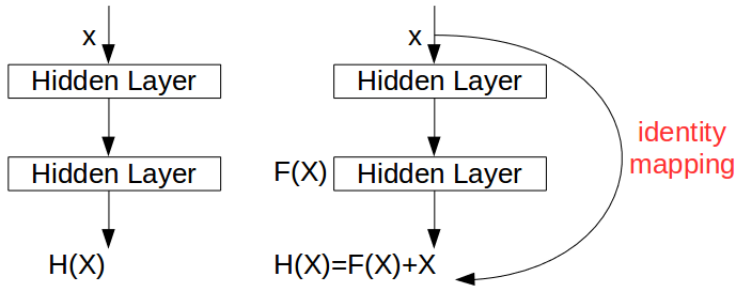


Figure 3.13: Left: Two Hidden Layers in Regular Neural Network. Right: Identity Connection is Skipping Two Hidden Layers. Figure is based on He et al. (2016).

where the identity mapping is skipping two layers.

From a practical perspective it must be mentioned that $F(\mathbf{X})$ and $\mathbf{X}$ are matrices and must have matching dimensions in order to be concatenated. Thus, the following options are available:

- All layers have the same number of units.
- A convolution layer is used to down- or upsample the input to match the dimensions.
- Some form of dropout is used to randomly select matching dimensions of the input matrix.

4

Related Work

The traditional approach to *HAR* involves the following chain,

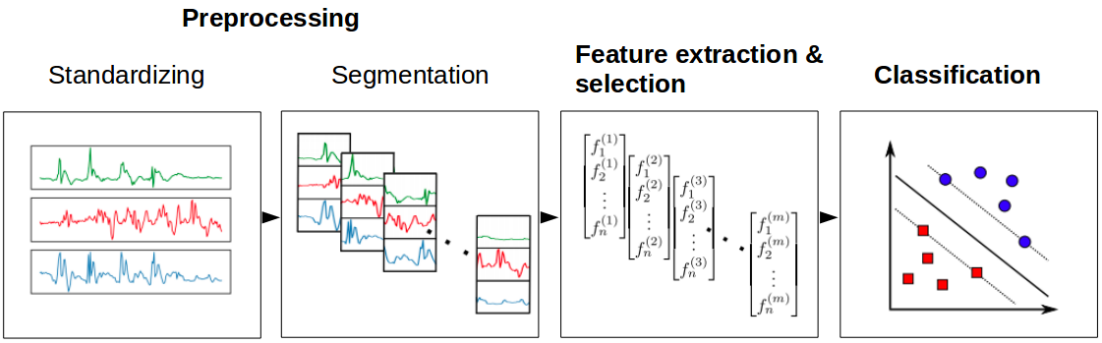


Figure 4.1: HAR Processing Chain. Figure is based on Li et al. (2018).

where the sensor recordings are aligned, standardized and segmented into windows

during the preprocessing stage. The label of each window is determined by some logic such as majority vote. After the segmentation stage, a feature extraction and selection strategy is deployed on each window to condense the raw signal into informative handcrafted features. The last stage then uses the extracted features as input for a classification algorithm (Bulling et al., 2014).

The feature extraction process can involve a wide range of handcrafted features such as:

- *Statistical features*: these are typical time and frequency domain features, which are popular because of their computational simplicity and their good performance across many activities (Ravi et al., 2005), (Bao & Intille, 2004).
- *Body structure features*: prior knowledge about the physiological body structure is modeled. Polynomial features such as slope and curvature are computed to model the trajectory of the limbs. Encoding physiological constraints and behaviour can increase the classification performance and help with the recognition robustness across subjects as shown by Zinnen et al. (2009).
- *Meta features*: an unsupervised algorithm like k-means (clustering) can be used on the raw signal or the handcrafted features to extract meta features, such as occurrence statistics, durations and distance scores (Huynh et al., 2008), (Blanke & Schiele, 2009), (Zhang & Sawchuk, 2012).

The feature selection procedure tries to find the most informative features so that the *HAR* task can be solved with minimum resources regarding computational effort and training samples. A variety of automatic selection procedures were proposed such as:

- *Wrapper approach*: the optimal feature subset is determined by repeatedly applying and evaluating the classification algorithm on different feature subsets. Typical approaches are forward selection, backward elimination and genetic selection (Kohavi & John, 1997).
- *Filter approach*: this strategy is typically based on statistical tests, feature correlations or criteria based on information theory (Peng et al., 2005).
- *Hybrid approach*: a mixture between the first two feature selection procedures (Somol et al., 2006).

The choice of classification algorithm can be critical for the success of the *HAR* chain, if the feature extraction and selection stage did not reveal well distinguishable patterns. A variety of methods were proposed. For example, a *support vector machine* (*SVM*) classifier was used by Bulling & Roggen (2011) to classify office activities, a *conditional random field* (*CRF*) to recognize composite activities by Blanke & Schiele (2010) and a decision tree by Bao & Intille (2004).

Current state-of-the-art results achieved with a traditional *HAR* chain involve among others the following papers:

- Zhu et al. (2017) use a random forest in combination with statistical features on the REALDISP dataset (ideal-placed scenario) to achieve a 99% weighted F1-score for a 10-fold cross validation.
- Baldominos et al. (2017) deploy a random forest mainly based on the fourier transformation to get a weighted F1-score of 97% on the PAMAP2 dataset for a train-test split evaluation. The focus of the paper is on the application of a genetic algorithm for feature selection.
- Chowdhury et al. (2018) construct an ensemble of *SVM*, *random forest* (*RF*) and many more methods, which are trained on statistical features, to achieve a weighted F1-score of 91% on the MHEALTH dataset in a train-test split evaluation.

In the last few years *NN*s started to achieve state-of-the-art results in several domains including *HAR*. The following listing presents a small selection of relevant papers:

- San et al. (2017) use a temporal *CNN* with two fully connected feedforward layers on top to process the raw signals and achieve an classification accuracy of 90% on the REALDISP dataset (self-displaced scenario) in a train-test split evaluation. However, the sets are not split along subjects.
- Ordóñez & Roggen (2016) deploy a hybrid neural network, consisting of temporal convolution layers for feature extraction on the raw signals and recurrent *LSTM* layers on top, to achieve 91% weighted F1-score on the gesture recognition task of the OPPORTUNITY challenge.
- Hammerla et al. (2016) evaluate a variety of *NN* architectures on several *HAR* datasets including DAPHNET. On the latter dataset, their best performing model (*LSTM*) achieves an 76% unweighted F1-score in a train-test split evaluation. It is further noteworthy that they introduce a novel carry-over probability for the memory states. Thus, the memory states in the next batch are initialized by the states in the previous batch with some probability. The states are reset to zero otherwise. This can be seen as a novel approach to regularization of stateful recurrent networks.

5

Experiments and Results

5.1 Implementation Details

All experiments are carried out by an evaluation framework developed in python 2.7. This pipeline makes extensive use of the open source machine learning libraries *Scikit-Learn* (Pedregosa et al., 2011) and *Keras* (Chollet et al., 2015) with *CNTK* (Seide & Agarwal, 2016) as backend[1]. It must be noted that the first choice regarding the backend was *Tensorflow* (Abadi et al., 2015), because it is well established and often used in the deep learning community. However, during preliminary experiments it was apparent that the

[1]CNTK enables the user to execute extensive matrix operations with the *graphical processing unit* (*GPU*).

results were not reproducible due to an issue with the random generator seeds. This problem is rooted in the keras library and is not resolved yet. The progress on this issue can be tracked on the keras github page[2]. Further note, that this thesis is done in cooperation with the company *AGT International*, which does not allow the publication of the source code.

The experiments will cover all neural network types outlined in chapter 3.3. More specifically:

- *LSTM and GRU networks*: These models are applied as introduced in section 3.3.1.1 and 3.3.1.2 respectively. Thus, the number of states corresponds to the number of timesteps in the input data.
- *CONV_LSTM*: This model consists of convolutional layers with *LSTM* layers on top. In contrast to the *2D* convolutions introduced in section 3.3.2, only *1D* convolutions along the time dimension of the input data are used. Hence, the kernel size always stretches over all input dimensions and must only be defined w.r.t. the number of timesteps.
- *RES_LSTM*: *ResNet*s were originally applied by He et al. (2016) on convolutional layers and designed for architectures with many layers. In the scope of this thesis, the concept of residuals is applied on *LSTM* layers and is limited to a maximum of five layers due to computational resources.
- *RF*: This model represents the traditional approach on *HAR* and is used as a baseline for comparison with the *NN*s.

[2] https://github.com/keras-team/keras/issues/2280

5.2 Datasets

In total, one private and three public datasets are selected for the experiments. The private dataset was recorded and annotated during the year 2017 and is an ongoing effort to this day.

The first public dataset is referred to as *REALDISP* and contains fitness related activities, such as jogging, rope jumping and stretching exercises. Baños et al. (2012) recorded 33 activity classes at 50 Hz for 17 subjects. The sensors provide acceleration, gyroscope and magnetometer recordings in 3*D* and orientation measurements in the quaternion format. However, to ensure that the dataset is still processable by the given resources, only acceleration and gyroscope data is considered. Furthermore, *REALDISP* was originally collected to study the effect of sensor-displacement on activity recognition methods and therefore provides three different sensor placement scenarios. Due to time constraints, only the ideal scenario is considered for this thesis. The distribution of the activities in the dataset and the activity duration are visualized in figure B.1 and B.2. Please note that some activity classes are extremely underrepresented in the dataset such as activity 17, which corresponds only to 0.4% of all available samples in the dataset.

The *OPPORTUNITY Project* is the second public dataset and was introduced in 2011 as a benchmark dataset for *HAR*. A subset of this dataset was used to pose the *OPPORTUNITY Challenge* (Chavarriaga et al., 2013). This dataset focuses on typical daily activities and covers five *ADL*[3] runs and one drill[4] run for four subjects. The data was recorded by

[3]Subjects carry out activities following a predefined sequence.
[4]Subjects execute activities repeatedly.

a total number of 72 sensors at a sample rate of 30 Hz.
This work follows the challenge guidelines for the task B2, which includes the classification of gestures during a breakfast scenario. The task focuses on 17 activities with gestures like opening/closing the door and cleaning the table. Activity distribution and the corresponding activity durations can be seen in figure B.3 and B.4.

The last public dataset is called *DAPHNET* and concerns the recognition of *freeze of gait*, which is a physical side effect of the Parkinson's disease. Affected subjects suddenly are afraid of moving forward and completely freeze in their motion. The dataset incorporates data of 10 subjects. Each subject was equipped with 3*D* acceleration sensors on trunk, thigh and shank, which recorded at 64 Hz (Baechlin et al., 2009). The corresponding activity distribution and activity durations can be found in figure B.5 and B.6.

Finally, the private dataset is focused on *mixed martial arts* (*MMA*), which is a full-contact combat sport. The fighters are allowed to apply striking, throwing and grappling techniques. This implies that the fight can transition from a standing position (striking and kicking techniques) to a clinch position (wrestling and judo techniques) or a ground position (submission techniques). The objective is to correctly classify the position based on the input data from sensors placed on each wrist and the waist. The sensors record acceleration and gyroscope data in three dimensions. The dataset contains recordings of 2 sparring fights. Each bout involves a different pair of fighters and covers 3 rounds, 3 minutes each. Unfortunately, the third round of the first bout could not be used due to sensor issues (data loss during the recording). The dataset overview, the activity distribution and the activity durations are visualized in table B.1, figure B.7 and B.8 respectively.

5.3 Experimental Setup

5.3.1 Data Preparation

Initially, the data is split into *train, validation and test sets* along the subjects to ensure there is no data leakage. This means that a model is being trained, optimized and evaluated on different data and subjects. Therefore, the generalization capabilities can be measured more accurately since all models are subject independent (Kaufman et al., 2011). However, there are two exceptions: The OPPORTUNITY dataset is split according to the challenge guidelines, which require a subject depended model. Furthermore, the *MMA* dataset contains only four subjects and therefore is not split along the subjects to ensure the resulting datasets are not too small.

After the initial split, the data is standardized based on the train set to enforce that all variables are on the same scale and centered around zero. The mean and the standard deviation of the train set will be used to rescale the validation and test set. The standardization is carried out by the following transformation for each raw signal dimension,

$$x_{std} = \frac{x - \mu}{\sigma} \tag{5.1}$$

where μ is the mean and σ is the corresponding standard deviation. This preprocessing step is mandatory for neural networks. The mean subtraction makes sure that the network weights increase or decrease independently of each other. To see this effect, consider a case where the input is composed of only positive values. The weights would either increase or decrease together, since the weight update only depends on the sign of the *backpropagated* error signal. Furthermore, if the input features are on a different

scale, the magnitude of the features can bias the networks due to their nonlinear[5] transformations (LeCun et al., 1998).

The last step segments the data into windows, which can possibly overlap, to frame the activity recognition task as a classification problem. Each window is then assigned an activity label by considering the most frequent activity class inside the window.

5.3.2 Evaluation Strategy

As already mentioned in the previous section, a *train*, *validation and test split* is used to estimate the generalization abilities of the models. Additionally, a *leave-one-subject-out-cross-validation* (*LOSOCV*) evaluation is carried out to gauge the impact of the subjects on the model performance. The procedure uses a single subject for performance evaluation and trains the model on the remaining subjects. This procedure is repeated until every subject was used as a test set once. This further helps to assess the generalization capabilities of a model, especially in use cases where it is known that a wide range of different subjects can be involved.

The generalization error can be measured with several metrics, depending on the given dataset and its activity distribution. The basis for every performance analysis in a classification setting is the *confusion matrix*. This matrix compares the model prediction with the ground truth labels and displays all the possible outcomes. The following table describes the basic terminology for the binary case with two classes,

[5] Small changes in the input can yield big effects on the output if a nonlinear transformation is used.

		Predicted Class	
		P	N
Actual Class	P	**TP**	**FN**
	N	**FP**	**TN**

Table 5.1: Confusion Matrix Terminology

where *P* is the positive class and *N* is the negative class [6]. The confusion matrix will count a true positive (*TP*) outcome, if the prediction is positive as well as the ground truth, a false positive (*FP*), if the prediction is positive while the ground truth is negative, a false negative (*FN*) in the reverse case and a true negative (*TN*), if the prediction and the ground truth are the negative class (Fawcett, 2006). This logic can be easily generalized to multiple classes.

Based on the confusion matrix, several performance measures can be derived. A common metric is *accuracy*, which is defined as follows:

$$ACC = \frac{TP + TN}{P + N} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.2}$$

This metric computes the ratio between correct predictions and total number of samples. The resulting score is between zero and one, where one is the best performance. For

[6] The class names were chosen to smoothly introduce the terminology. They do not have any further meaning.

multiple classes, the metric is given by,

$$ACC = \frac{1}{C}\sum_{c} ACC_c = \frac{1}{C}\sum_{c=1}^{C} \frac{TP_c + TN_c}{TP_c + TN_c + FP_c + FN_c} \tag{5.3}$$

where the *accuracy* is computed for each class c separately and then averaged to get a final score. However, this metric is no reliable measure, if the dataset is highly imbalanced, which usually is the case in *HAR*. The null class[7] often is the most frequent class. Hence, a trivial classifier could simply predict the most frequent activity and achieve a reasonable performance, even though the classifier did not learn to distinguish the activities (Fawcett, 2006).

A more reliable alternative to the previous performance score is called *F1-Measure*, which is defined for binary classification tasks as follows:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{5.4}$$

This metric computes the harmonic mean between precision and recall, which are in turn given by:

$$Precision = \frac{TP}{TP + FP} \tag{5.5}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.6}$$

Precision measures how many correct predictions are made in relation to the total number of predictions for some class and *recall* measures the number of correct predictions in relation to the total number of samples of some class. The F1-Measure can be gener-

[7] None of the defined activities are executed.

alized to multiple classes in several ways; two very common approaches are given by,

$$F1_{macro} = \frac{1}{C}\sum_{c=1}^{C} F1_c = \frac{1}{C}\sum_{c=1}^{C} 2 \times \frac{Precision_c \times Recall_c}{Precision_c + Recall_c} \tag{5.7}$$

$$F1_{weighted} = \frac{1}{C}\sum_{c=1}^{C} \frac{N_c}{N} \times F1_c \tag{5.8}$$

where C is the number of classes, N_c is the number of samples in class c and N is the total number of samples in the dataset. While the *macro* variant is treating each class equally important by taking an unweighted average, the *weighted* version is weighting each class score by its corresponding sample size proportion (Fawcett, 2006).

5.3.3 Baseline Model

A *RF* in combination with several handcrafted features is used as a benchmark for the comparison with the neural networks. This comparison between a traditional machine learning approach and the more recent neural network methodology shall give further context to correctly asses the practicability of *NN*s for *HAR*.

RF was introduced by Breiman (2001) and is an ensemble method that combines several decision trees. Each tree is fitted on bootstrapped[8] data and is usually not regularized. Thus, every tree estimator exhibits a low bias and high variance in its expected generalization error. Hence, by averaging[9] the prediction of several trees, the variance can be decreased. Furthermore, since each tree will only consider a random subset of all features to create nodes, the ensemble of trees will be decorrelated, which in turn

[8] Random sampling with replacement (Hastie et al., 2001).

[9] Computing the mean prediction for regression tasks and taking the majority vote in classification problems.

further decreases the variance of the estimator. A more rigorous description of the random forest can be found in Breiman (2001) and Hastie et al. (2001).

The handcrafted features are typical time and frequency domain statistics, which are widely used in *HAR* (Bao & Intille, 2004). Each handcrafted feature is computed for every raw input dimension of a sliding window. A overview of all computed features can be found in appendix A.3. Further note that a feature selection strategy was not deployed since the number of features was not too excessive compared to the sample size. However, feature selection can in theory further improve the performance of the baseline classifier.

5.3.4 Hyperparameter Optimization

The validation set resulting from the *train, validation and test split* is used to find the best performing hyperparameters. It is critical to note that in both evaluation strategies the validation set was never combined with the train or test set. Furthermore, the decisive metric used on the validation set is the $F1_{macro}$. Therefore, all classes are treated as being equally important in contributing to the overall performance.

A simple grid search is executed to find the best performing model configurations. However, the grid range varies from dataset to dataset and is manually adjusted as soon as it is apparent that the current parameter grid is not leading to further performance improvements. For example, if the grid range for the number of units per layer is between 50 and 1000 and the performance started to decline after 400 units consistently, the grid search is canceled and restarted for a different hyperparameter. Due to time

and resource constraints, the *NN*s are optimized with focus on the following hyperparameters:

- Number of layers
- Type of layer
- Units per layer
- Batch size
- Dropout probability

Optimizing the learning rate is avoided by choosing an adaptive learning rate procedure called *Adam*[10] and tuning of the number of iterations is bypassed by using *early stopping*[11]. Further note that preliminary experiments suggested a slight performance increase, if recurrent layers pushed their output for all timesteps h_t to the next layer instead of just the last output h_T. In addition, *dropout*[12] regularization is applied on the input of each recurrent layer and the *Glorot* weight initialization scheme is used across all datasets. Furthermore, the default activation functions for the recurrent layers are used as described in section 3.3.1 as well as the *ReLu* activation for convolution layers.

It is important to mention that the random generator seeds are fixed for the network initialization and the data preparation (shuffling the sliding windows) to ensure reproducibility. This has a potential downside for the *NN*s, since they could be initialized with disadvantageous weights by chance. However, the alternative would involve repeatedly evaluating every network architecture to see the impact of the initialization scheme. This approach allows more insight to the generalization capabilities but is too

[10] Adam was introduced in section 3.2.6.2.
[11] Early stopping was described in section 3.2.7.
[12] Dropout was described in section 3.2.7.

time consuming for the given resources.

Lastly, the baseline estimator is optimized w.r.t. the number of trees, the tree depth and the size of the feature subset. While the number of trees and the feature subset size directly influence the variance of the estimator, the tree depth impacts the bias of the estimator.

5.4 Results and Discussion

5.4.1 REALDISP

In this experiment, the data is split into a training set consisting of subjects 10-13 and 6-9, a validation set containing subjects 4,5 and 14 and a test set involving subjects 1-3 and 15-17. The data is segmented by a sliding window with a length of 3 seconds and a stride of 0.5 seconds. Further note that all neural networks are trained with a batch size of 32.

The best performing models for each type are detailed in table B.2 and the result for the *train, validation and test split* evaluation is listed in the following table:

Model / Metric	**Accuracy**	**F1 Macro**	**F1 Weighted**
RF	**0.93823**	**0.88465**	**0.93915**
LSTM	0.92939	0.87170	0.93037
GRU	0.92122	0.83153	0.92443
CONV_LSTM	0.92868	0.87045	0.93072
RES_LSTM	0.92002	0.84513	0.91905

Table 5.2: REALDISP: Performance Metrics for Train-Validation-Test Split.

The results from the above table show that all models achieve a very similar perfor-

mance in accuracy and F1 weighted. However, *GRU* and *RES_LSTM* underperform with respect to the F1 macro metric, while the *RF* shows the best performance in all metrics. It is particularly interesting to investigate the confusion matrix of the *RF* in figure B.10. Most classes are recognized very well, however label 17 (repetitive forward stretching) is often confused with label 11 (waist bends forward) and label 1 (walking) with label 0 (null class). The first case is no surprise since both exercises are indeed similar or can at least be executed with a lot of similar movements. The latter case is also expected since the subjects were most likely walking around between the activities. An unexpected confusion occurs between label 16 (lateral bend arm up) and label 28 (Knees bending). A possible explanation is related to the fact that most sensors were placed on the upper body. Further hypothesizing that some subjects were executing the activities with similar arm movements, a confusion would be expected to some degree.

It can be further hypothesized that the small number of activity samples for some classes affect the performance of the neural networks more severely than the performance of the baseline classifier. This seems plausible since *NN*s need to estimate a higher amount of parameters during the training procedure as compared to the baseline model.

The result for the *LOSOCV* evaluation is visualized in figure B.9 and the corresponding means and standard deviations are presented in the following table:

Model / Metric	Accuracy	F1 Macro	F1 Weighted
RF	**0.94419**±0.02736	**0.86924**±0.11407	**0.94323**±0.02890
LSTM	0.93420±0.03020	0.86201±0.07196	0.93380±0.03462
GRU	0.91839±0.03520	0.81424±0.09686	0.91922±0.03920
CONV_LSTM	0.93378±0.03151	0.85130±0.08615	0.93145±0.04252
RES_LSTM	0.92802±0.03453	0.84039±0.08105	0.92598±0.04179

Table 5.3: REALDISP: LOSOCV Performance.

This result is very similar to the previous one, with *RF* being the best performing model. However, considering the *interquartile range* (*IQR*) shown in the visualization B.9, it is apparent that the model performance varies across the subjects significantly. This is especially noticeable in the F1 macro metric, where all estimators have a large *IQR* except the *LSTM*. Therefore, the *LSTM* might be preferred to the *RF* even though it has a lower mean performance.

5.4.2 OPPORTUNITY

This dataset is split according to the challenge guidelines. Therefore, the train set contains all recordings for subject 1 and *ADL*1, *ADL*2 and the drill recording for subject 2-3. The validation set is involving the third *ADL* recording for subject 2 and 3 and the test set contains *ADL*4 and *ADL*5 recordings of subject 4 and 5. A sliding window segments the data into 0.8 second parts with an overlap of 50% and the batch size for the training of the neural networks is 64.

The best performing models for each type are detailed in table B.3 and the results for the *train, validation and test split* evaluation are listed in the following table:

Model / Metric	Accuracy	F1 Macro	F1 Weighted
RF	0.88666	0.44888	0.86321
LSTM	0.87597	0.58987	0.87800
GRU	0.86733	0.55332	0.87062
CONV_LSTM	**0.89317**	**0.63828**	**0.89561**
RES_LSTM	0.87679	0.57304	0.88017
CNN (Ordóñez & Roggen, 2016)	-	0.535	0.883
DeepConvLSTM (Ordóñez & Roggen, 2016)	-	**0.704**	**0.915**

Table 5.4: OPPORTUNITY: Performance Metrics for Train-Validation-Test Split.

The result clearly displays the *CONV_LSTM* as the best performing model out of all the evaluated candidates. Examining the corresponding confusion matrix in figure B.19, it is apparent that the main issue is the null class. Almost all classes are affected by this confusion.

A particularly interesting comparison occurs between *CONV_LSTM* and *DeepConvLSTM* (Ordóñez & Roggen, 2016). They use the same network type (1*D* convolution layers with *LSTM* layers on top), but achieve a much better result with respect to the F1 metric. This can be explained partly by a different random seed and a different weight initialization strategy. However, a considerable amount of time was invested in reproducing their result. The reproduction was unsuccessful and since they did not mention how many training iterations were used, this is most likely the reason the reproduction failed. Further note that the F1 macro scores of Ordóñez & Roggen (2016) are reconstructed based on their reported confusion matrices.

The results for the *LOSOCV* evaluation is visualized in figure B.15 and the corresponding means and standard deviations are presented in the next table:

Model / Metric	Accuracy	F1 Macro	F1 Weighted
RF	0.77050±0.03797	0.32024±0.09153	0.71192±0.06198
LSTM	0.71688±0.10298	0.41066±0.13842	0.71436±0.08203
GRU	**0.75812**±0.06965	**0.44969**±0.10153	**0.74864**±0.06439
CONV_LSTM	0.75488±0.06789	0.41671±0.07954	0.72976±0.05342
RES_LSTM	0.71539±0.11314	0.41862±0.13112	0.71420±0.08478

Table 5.5: OPPORTUNITY: LOSOCV Performance.

The *LOSOCV* evaluation shows a significant performance drop in all metrics for all models. This indicates that the task is much more difficult for subject independent models, which is the more likely case in real life applications. The *GRU* shows a superior performance w.r.t. all metrics, however this evaluation contains only three[13] different subjects. Hence, the *LOSOCV* result should be interpreted with care.

5.4.3 DAPHNET

In this experiment, the train set contains all recordings except from subjects 2 and 9. The validation set is based on subject 9 and the remaining recordings from subject 2 are used as a test set. The sets are segmented into 50% overlapping windows with a length of 1 second and the neural networks are trained with a batch size of 128.

The best performing models for each type are detailed in table B.4 and the results for the *train, validation and test split* evaluation are listed in the following table:

[13]The remaining subject was selected to be the validation set.

Model / Metric	Accuracy	F1 Macro	F1 Weighted
RF	0.88971	0.61069	0.85569
LSTM	**0.92647**	0.78848	0.91548
GRU	0.88971	0.71169	0.88010
CONV_LSTM	0.91728	0.79159	0.91190
RES_LSTM	0.92002	**0.84513**	**0.91905**
CNN (Hammerla et al., 2016)	-	0.684	-
LSTM (Hammerla et al., 2016)	-	0.760	-

Table 5.6: DAPHNET: Performance Metrics for Train-Validation-Test Split.

The result indicates the *RES_LSTM* is the best model w.r.t. the F1 scores. This result is particularly clear in case of the F1 macro metric, where the *RES_LSTM* shows a performance 5% better than the second best model and roughly 12% better than the result reported by Hammerla et al. (2016). It is further insightful to compare the confusion matrices of the *RES_LSTM* and the *RF* in figure B.26 and figure B.22 respectively. While the *RF* fails to recognize the activity, the *RES_LSTM* learns to distinguish the class at least to some degree.

From a health care perspective, the recognition of the freeze of gait is crucial and more important than the identification of the null class. Therefore, the *RF* cannot be used even though the evaluation metrics indicate a decent performance. It must be further noted that in this context, the F1 macro score is not an appropriate decision function for hyperparameter optimization. The resulting model should focus on maximizing the true positives of the freeze of gait class while keeping the false positives at an acceptable level.

The results for the *LOSOCV* evaluation is visualized in figure B.21 and the correspond-

ing means and standard deviations are presented in the next table:

Model / Metric	Accuracy	F1 Macro	F1 Weighted
RF	**0.89737**±0.09349	0.58238±0.17773	**0.86102**±0.13250
LSTM	0.86741±0.11985	0.61619±0.17732	0.85034±0.13473
GRU	0.79045±0.19877	0.56812±0.20029	0.78980±0.17413
CONV_LSTM	0.84436±0.10128	**0.63511**±0.17268	0.85246±0.10175
RES_LSTM	0.87942±0.08927	0.61849±0.16266	0.86007±0.12442

Table 5.7: DAPHNET: LOSOCV Performance.

The *LOSOCV* result suggests, just like in the previous two datasets, a strong impact of the subjects on the classification performance. The *RF* shows the best performance for the accuracy score and the F1 weighted metric. However, examining the averaged and normalized confusion matrix in figure B.27, it is obvious that this model is unable to distinguish the classes.

Considering the F1 macro score, the *CONV_LSTM* is the best performing model, followed by the *RES_LSTM* and the *LSTM*. Further taking the *IQR* in figure B.21 into account, it can be argued that the *RES_LSTM* exhibits a more robust performance across subjects, even though the mean performance is slightly below the *CONV_LSTM*.

5.4.4 MMA

The *mixed martial arts* (*MMA*) dataset is split into a train set containing the first bout, a validation set consisting of the first round of the second bout and a test set with the remaining rounds. A sliding window segments the data into 0.6 second parts with 50% overlap. Further note that the neural networks are trained with a batch size of 32.

The best performing models for each type are detailed in table B.5 and the results for the *train, validation and test split* evaluation are listed in the following table:

Model / Metric	Accuracy	F1 Macro	F1 Weighted
RF	0.65931	0.47965	0.58575
LSTM	0.67937	0.58561	0.65499
GRU	0.66659	0.57397	0.64171
CONV_LSTM	**0.69118**	**0.62787**	**0.68679**
RES_LSTM	0.69060	0.56046	0.64578

Table 5.8: MMA: Performance Metrics for Train-Validation-Test Split.

The result indicates that the *CONV_LSTM* is the best performing model across all metrics. The corresponding confusion matrix in figure B.32 shows that the main confusion originates from the clinch position, which is often confused with the standing position. This is an expected outcome, since both positions require an upright fighting stance and contain similar movements like uppercuts and hooks. However, the standing position is rarely confused with the clinch position. Therefore, it can be hypothesized that the standing position provides sufficient pattern in the data to be identified, while the clinch position resembles the standing position but lacks a clear identification pattern.

The result of the *LOSOCV* evaluation is visualized in figure B.28 and the corresponding means and standard deviations are presented in the next table:

Model / Metric	Accuracy	F1 Macro	F1 Weighted
RF	0.76619±0.09842	0.55648±0.03423	0.75590±0.12670
LSTM	**0.77348**±0.08053	0.57980±0.07029	**0.77502**±0.10105
GRU	0.76090±0.09252	0.56141±0.07185	0.76076±0.11544
CONV_LSTM	0.73416±0.023129	0.58099±0.12357	0.75877±0.05400
RES_LSTM	0.76913±0.08233	**0.58483**±0.11577	0.77305±0.10376

Table 5.9: MMA: LOSOCV Performance.

Similar to the *LOSOCV* evaluation of the OPPORTUNITY dataset, the above result should be interpreted with care, since the *MMA* dataset contains only four different subjects. The result shows an increased performance in accuracy and F1 weighted compared to the the *train, validation and test split* evaluation. This can be attributed to the different activity distributions associated with the left out subjects. It can be further noted that the *LSTM* displays the best performance in accuracy and F1 weighted, while the *RES_LSTM* is the best model with regards to the F1 macro. Considering the *IQR* range of the models and the corresponding outliers displayed in figure B.28, it is apparent that a conclusive interpretation is difficult. The *RF* has the smallest *IQR* and might be the most robust with respect to the subjects. However, it also has the smallest mean performance in F1 macro. The *RES_LSTM* has the best mean F1 macro performance, although this can be explained by the outlier shown in the visualization. Therefore, the *LSTM* seems to be a suitable compromise between robustness and mean performance with respect to all metrics.

6

Conclusion and Future Work

6.1 Conclusion

This thesis starts with a survey of past and current research approaches on *HAR* with wearable sensors. The main contribution of the present work involves an exhaustive empirical evaluation of different *RNN* based models to recognize a wide range of activities. The most noteworthy approach is the *ResNet*. This method is a novel approach in sensor based *HAR*, because it is evaluated on a small scale and in combination with recurrent layers.

The *NN*s are further compared with a traditional machine learning approach, which

depends on the computation of handcrafted features. These features often require specific domain knowledge and additional computational resources. The latter is especially problematic in the case of a real-time application, where the prediction time can be critical.

The empirical evaluation is based on four different datasets, covering a wide range of activities, subjects and sensor setups. The results strongly indicate that the proposed networks are at least competitive with the traditional approach. In three out of four datasets, the *NN*s achieve a superior performance in both evaluation setups. Thus, the proposed networks can be used to effectively avoid feature engineering, while achieving a competitive performance. However, the hyperparameter optimization procedure for the *RNN*s required a lot of effort, because *RNN*s contain more hyperparameters than most traditional machine learning models. Hence, even if the practitioner can save time by avoiding handcrafted features, he most likely will spend additional effort on hyperparameter optimization.

In conclusion, *RNN*s are suitable to tackle sensor based *HAR* problems, considering the results of the empirical evaluation. This statement holds in particular, if there is a sufficient amount of data available and if the practitioner lacks domain specific knowledge.

6.2 Future Work

There are at least three main research directions for future work. The first one involves a specific implementation property of *RNN*s known as statefulness. In the current *Keras*

implementation, the memory states of *RNN*s are reseted after each batch by default and therefore are not stateful. Although, it might be useful to use the last memory states from the previous batch to initialize the states of the next batch. This property could help to avoid the prediction of activities that are very unlikely given the previous predictions. However, it must be mentioned that states do not necessarily contain enough information to indicate a certain activity.

Another research possibility is concerning post processing methods. A median filter could be used to smooth out the predicted activities in a given time window. *CRF*s could be applied on the network output to model the transition probabilities between the activities. Both methods help to identify very unlikely activities given the previous prediction. Hence, combining stateful networks with appropriate post processing methods has the potential to increase the performance considerably.

The last potential research direction focuses on the way the *HAR* task is framed. All deployed models are trained in a supervised fashion, depending on correctly labeled activities. However, it often is very costly to annotate a recorded dataset manually, especially if the recorded activities are not clearly defined or are hard to identify. Therefore, it might be more efficient to focus on unsupervised techniques or at least combine supervised with unsupervised methods, so that only a small amount of labeled data is necessary.

A

Mathematical Derivations

A.1 Weight Initialization Scheme

The following derivation is based on chapter 7 in Reed & Marks (1999).

Recalling the linear combination of all incoming connections of a single hidden unit as shown in equation 3.5 and neglecting the bias term. Further assuming the weights have

a zero mean and are *IID*, the following properties can be derived for a specific weight w_i,

$$E[w_i x_i] \overset{\text{IID}}{=} E[w_i]E[x_i] = 0 \tag{A.1}$$

$$E[w_i] = 0 \tag{A.2}$$

$$E[w_i^2] = \sigma_w^2 \tag{A.3}$$

where σ_w^2 is the variance of the population of weights w, following some distribution. Further assuming the inputs of the specific hidden unit are zero mean and *IID*,

$$z_k = \sum_{i=1}^{N} w_i x_i \tag{A.4}$$

$$E[z_k] \overset{\text{IID}}{=} \sum_{i=1}^{N} E[w_i]E[x_i] = 0 \tag{A.5}$$

$$Var[z_k] = E[z_k^2] = E[\big(\sum_{i=1}^{N} w_i x_i\big)^2] \overset{\text{IID}}{=} \sigma_w^2 \sum_{i=1}^{N} E[x_i^2] \tag{A.6}$$

$$Var[z_k] = \sigma_z^2 = N_{in}\sigma_w^2\sigma_x^2 \tag{A.7}$$

$$\sigma_z = \sqrt{N_{in}}\sigma_w\sigma_x \tag{A.8}$$

where N_{in} is the number of incoming connections.

Specifying the activation function for a specific hidden unit h_k, e. g. tanh, yields:

$$h_k = tanh(z_k) \tag{A.9}$$

$$z_k = \ln\left(\frac{1+h_k}{1-h_k}\right) \tag{A.10}$$

According to figure 3.4, the activation function saturates approximately at $|h_k| > 0.9$. This means that the first-order derivative evaluated at values satisfying this range will lead to very small output. Thus, plugging in the saturation value yields:

$$z_k^{sat} = \ln(19) \tag{A.11}$$

In order to enforce $P(z_k > z_k^{sat}) < \varepsilon$, the weight distribution can be constructed such that z_k^{sat} is drawn from a Gaussian distribution and is a multiple of the standard deviation σ_z:

$$z_k^{sat} > 2\sigma_z \tag{A.12}$$

$$z_k^{sat} > 2\sigma_w\sigma_x\sqrt{N_{in}} \tag{A.13}$$

Solving for σ_w and plugging in equation A.11 gives:

$$\sigma_w < \frac{z_k^{sat}}{2\sigma_x\sqrt{N_{in}}} = \frac{\ln(19)}{2\sigma_x\sqrt{N_{in}}} \tag{A.14}$$

Assuming the tanh activation is used for the specific unit, the weights could be initialized

by drawing from a Gaussian distribution such as:

$$w_i \sim \mathcal{N}\left(0, \sigma_w^2 < \frac{\ln(19)^2}{2\sigma_x^2 N_{in}}\right) \tag{A.15}$$

A.2 Backprop Derivation for LSTM

The following derivation is based on Graves (2008) and Greff et al. (2017).

The starting point of the *backprop* derivation is the formal definition of a *LSTM* cell at timestep t:

$$\mathbf{a}_t = \sigma_{tanh}\left(\mathbf{W}_a \mathbf{x}_t + \mathbf{U}_a \mathbf{h}_{t-1} + \mathbf{b}_a\right) \tag{A.16}$$

$$\mathbf{i}_t = \sigma_{sigm}\left(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i\right) \tag{A.17}$$

$$\mathbf{f}_t = \sigma_{sigm}\left(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f\right) \tag{A.18}$$

$$\mathbf{o}_t = \sigma_{sigm}\left(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o\right) \tag{A.19}$$

$$\mathbf{s}_t = \mathbf{a}_t \odot \mathbf{i}_t + \mathbf{f}_t \odot \mathbf{s}_{t-1} \tag{A.20}$$

$$\mathbf{h}_t = \sigma_{tanh}(\mathbf{s}_t) \odot \mathbf{o}_t \tag{A.21}$$

where σ is applied element-wise and $\odot$ is the *Hadamard product*. The first-order deriva-

tive of the objective function w.r.t. $\mathbf{h}_t$ can be defined as follows,

$$\delta \mathbf{h}_t \equiv \frac{\partial \mathbf{C}(\mathbf{Y}, \mathbf{T})}{\partial \mathbf{h}_t} \tag{A.22}$$

where $\mathbf{Y}$ is the final network prediction and $\mathbf{T}$ is the target variable. Now, by applying the chain rule, the first-order derivatives w.r.t. the state $\mathbf{s}_t$ and the output gate $\mathbf{o}_t$ are given as:

$$\delta \mathbf{s}_t = (1 - \sigma_{tanh}(\mathbf{s}_t)^2) \odot \mathbf{o}_t \odot \delta \mathbf{h}_t \tag{A.23}$$

$$\delta \mathbf{o}_t = \sigma_{tanh}(\mathbf{s}_t) \odot \delta \mathbf{h}_t \tag{A.24}$$

We can further derive the partial derivative for all the terms in equation A.20 as follows

$$\delta \mathbf{a}_t = \mathbf{i}_t \odot \delta \mathbf{s}_t \tag{A.25}$$

$$\delta \mathbf{i}_t = \mathbf{a}_t \odot \delta \mathbf{s}_t \tag{A.26}$$

$$\delta \mathbf{f}_t = \mathbf{s}_{t-1} \odot \delta \mathbf{s}_t \tag{A.27}$$

$$\delta \mathbf{s}_{t-1} \mathrel{+}= \mathbf{f}_t \odot \delta \mathbf{s}_t \tag{A.28}$$

where the last equation can be combined with equation A.23 to get:

$$\delta \mathbf{s}_t = (1 - \sigma_{tanh}(\mathbf{s}_t)^2) \odot \mathbf{o}_t \odot \delta \mathbf{h}_t + \mathbf{f}_{t+1} \odot \delta \mathbf{s}_{t+1} \tag{A.29}$$

Furthermore, since the partial derivatives w.r.t. the input weights are shared over time they are given by,

$$\delta \mathbf{W_a} = \sum_t (1 - \mathbf{a_t^2}) \odot \mathbf{x}_t \odot \delta \mathbf{a}_t \tag{A.30}$$

$$\delta \mathbf{W_i} = \sum_t \mathbf{i}_t \odot (1 - \mathbf{i_t}) \odot \mathbf{x}_t \odot \delta \mathbf{i}_t \tag{A.31}$$

$$\delta \mathbf{W_f} = \sum_t \mathbf{f}_t \odot (1 - \mathbf{f_t}) \odot \mathbf{x}_t \odot \delta \mathbf{f}_t \tag{A.32}$$

$$\delta \mathbf{W_o} = \sum_t \mathbf{o}_t \odot (1 - \mathbf{o_t}) \odot \mathbf{x}_t \odot \delta \mathbf{o}_t \tag{A.33}$$

and the partial derivatives w.r.t. the recurrent weights,

$$\delta \mathbf{U_a} = \sum_t (1 - \mathbf{a_t^2}) \odot \mathbf{h}_{t-1} \odot \delta \mathbf{a}_t \tag{A.34}$$

$$\delta \mathbf{U_i} = \sum_t \mathbf{i}_t \odot (1 - \mathbf{i_t}) \odot \mathbf{h}_{t-1} \odot \delta \mathbf{i}_t \tag{A.35}$$

$$\delta \mathbf{U_f} = \sum_t \mathbf{f}_t \odot (1 - \mathbf{f_t}) \odot \mathbf{h}_{t-1} \odot \delta \mathbf{f}_t \tag{A.36}$$

$$\delta \mathbf{U_o} = \sum_t \mathbf{o}_t \odot (1 - \mathbf{o_t}) \odot \mathbf{h}_{t-1} \odot \delta \mathbf{o}_t \tag{A.37}$$

and w.r.t. the bias vectors:

$$\delta \mathbf{b_a} = \sum_t (1 - \mathbf{a_t^2}) \odot \delta \mathbf{a}_t \tag{A.38}$$

$$\delta \mathbf{b_i} = \sum_t \mathbf{i}_t \odot (1 - \mathbf{i_t}) \odot \delta \mathbf{i}_t \tag{A.39}$$

$$\delta \mathbf{b_f} = \sum_t \mathbf{f}_t \odot (1 - \mathbf{f_t}) \odot \delta \mathbf{f}_t \tag{A.40}$$

$$\delta \mathbf{b_o} = \sum_t \mathbf{o}_t \odot (1 - \mathbf{o_t}) \odot \delta \mathbf{o}_t \tag{A.41}$$

Further note that the gradient for $\delta \mathbf{h}_{t<T}$ can be propagated over a variety of paths. Thus, the derivative w.r.t. the cell output can be defined as,

$$\delta \mathbf{h}_t = (1 - \mathbf{a}_{t+1}^2) \odot \mathbf{U}_a \odot \delta \mathbf{a}_{t+1} \tag{A.42}$$

$$+ \mathbf{i}_{t+1} \odot (1 - \mathbf{i}_{t+1}) \odot U_i \odot \delta \mathbf{i}_{t+1} \tag{A.43}$$

$$+ \mathbf{f}_{t+1} \odot (1 - \mathbf{f}_{t+1}) \odot U_f \odot \delta \mathbf{f}_{t+1} \tag{A.44}$$

$$+ \mathbf{o}_{t+1} \odot (1 - \mathbf{o}_{t+1}) \odot U_f \odot \delta \mathbf{o}_{t+1} \tag{A.45}$$

$$+ \mathbf{V} \odot \delta \mathbf{Y}_t \tag{A.46}$$

where the last part A.46 of the above equation is the *backpropagated* error signal from network output at the current timestep. This term will be zero if the current timestep is not connected to the output layer of the network.

Finally, the gradient expressions for all the weights in a *LSTM* cell can be used in combination with some optimization procedure to update the weights.

A.3 Handcrafted Features

Assuming an input matrix **X** of dimension $\mathbb{R}^{J \times M}$ with J samples and M raw features, where a single feature could be for example the acceleration in some direction. Each handcrafted feature is computed per raw feature dimension for every sliding window unless stated otherwise. The following table gives an overview of all computed features:

Table A.1: Handcrafted Features.

Feature	**Description**
MeanAD, *MedianAD*	Mean and median absolute deviation.
Min, *Max*	Minimum and maximum values.
Energy	Secord-order uncentered moment.
Kurtosis	Quantifies the amount of outliers.
Skewness	Measures the asymmetry of the probability distribution.
Autocorrelation coefficients	Measures the periodicity of a signal.
Correlation coefficients	Correlation between the raw feature dimensions.
Quantiles	Computes 5*th*, 25*th*, 75*th* and 90*th* quantiles.
IQR	Interquartile range.
Sign changes	Counts the number of zero crossings.
Peaks	Number of peaks.
Distance between peaks	Mean distance between peaks.
Signal magnitude	Mean L_2 norm across all raw features.
Entropy	Measures the randomness in a raw feature.
Fourier transform	The five biggest coefficients of the fourier series.

The reader is expected to be familiar with typical statistical measurements like *IQR*, *min*, *max*, *kurtosis*, *skewness*, *correlation* and *quantiles*. Therefore, only handcrafted fea-

tures will be exemplified, which are not self-explanatory or not well known within the area of statistics:

MeanAD and *MedianAD* are given as follows,

$$MeanAD = \sqrt{\sum_j \mid x_j - \bar{x} \mid} \tag{A.47}$$

$$MedianAD = \sqrt{\sum_j \mid x_j - \tilde{x} \mid} \tag{A.48}$$

where $\bar{x}$ and $\tilde{x}$ are the mean and median respectively.

Entropy is a concept from information theory and is quantifying the uncertainty in the data,

$$H = -\sum_j p(x_j) \ln\big(p(x_j)\big) \tag{A.49}$$

where $p(x)$ is the probability distribution of x. Please note that if $p(x) \in [0,1]$ then $-p(x)\ln\big(p(x)\big) \in [0,1]$. Furthermore, the maximum uncertainty is reached if and only if all probabilities are equal, $p_1 = \cdots = p_J = \frac{1}{J}$. Thus, entropy is bounded as follows:

$$0 \leq H \leq \ln(J) \tag{A.50}$$

A more detailed introduction can be found in chapter 1.6 of Bishop (2006).

The *discrete fourier transformation* (*DFT*) assumes that x_j is a periodical signal and converts it from the time domain into the frequency domain by decomposing the signal into a finite sum of sinusoidal functions:

$$x_j = \sum_{k=0}^{J-1} c_k e^{i2\pi kj/J} \qquad for \quad k = 0, 1, ..., J-1 \tag{A.51}$$

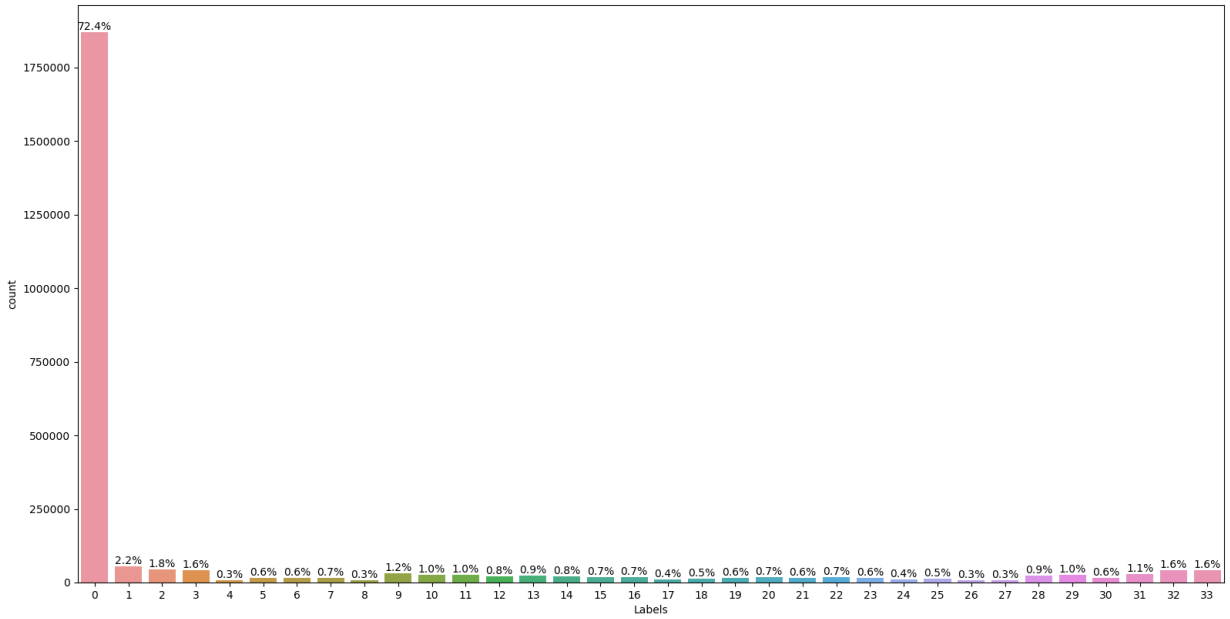
where c_k are the series coefficients. The explicit expression for the coefficients can be stated as follows:

$$c_l = \frac{1}{J} \sum_{j=0}^{J-1} x_j e^{-i2\pi lj/J} \qquad for \quad l = 0, 1, ..., J-1 \tag{A.52}$$

A rigorous introduction to frequency analysis and the fourier series in particular can be found in chapter 4 of Proakis & Manolakis (2006).

B

Figure B.1: Activity Distribution for the Ideal Scenario. The class mapping can be found in Baños et al. (2012).



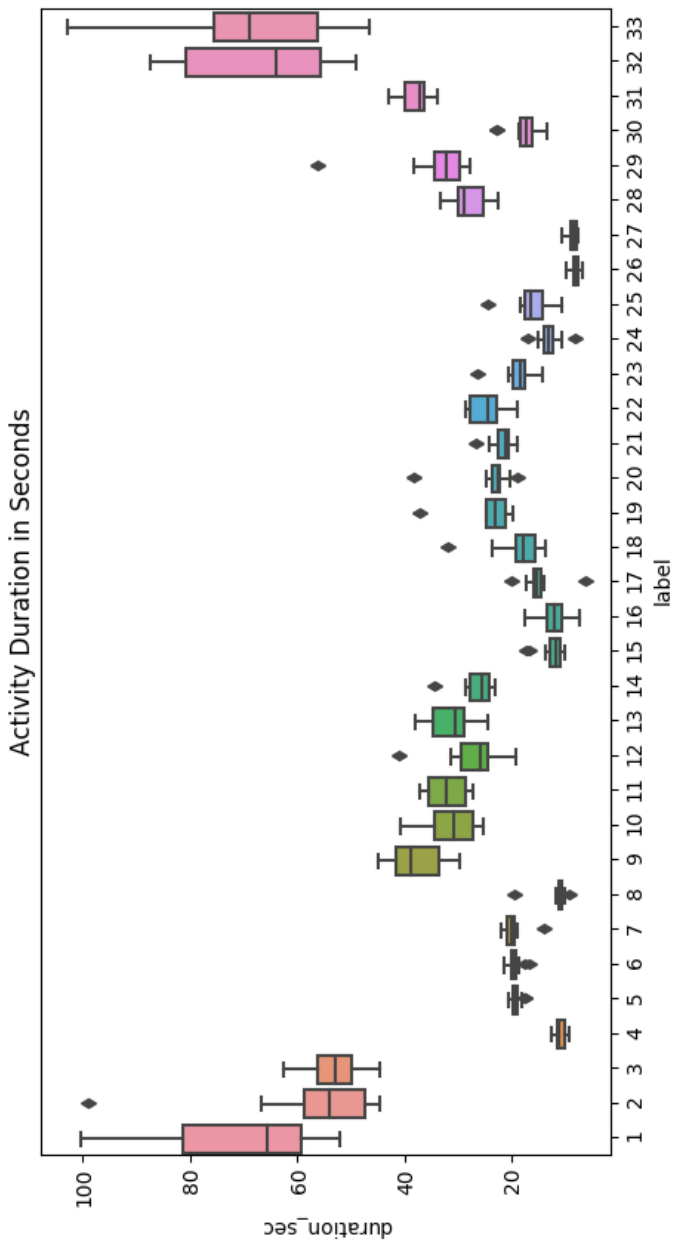


Figure B.2: Activity Duration in Seconds for the Ideal Scenario. The class mapping can be found in Baños et al. (2012).

Additional Figures and Tables

B.1 Dataset Details

Figure B.3: Activity Distribution including the Null-Class. The class mapping can be found in Chavarriaga et al. (2013).

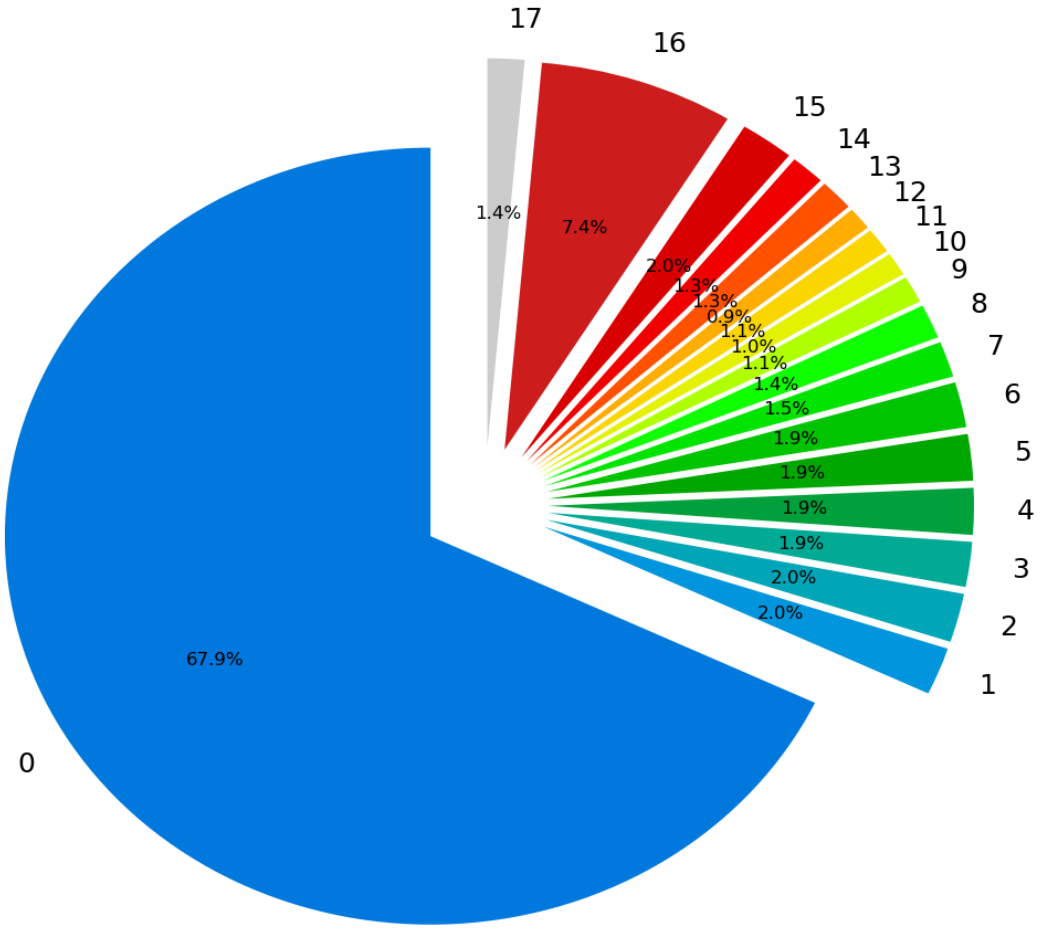


Figure B.5: Activity Distribution including the Null-Class. The 'Freeze of Gait' activity is mapped as 1.

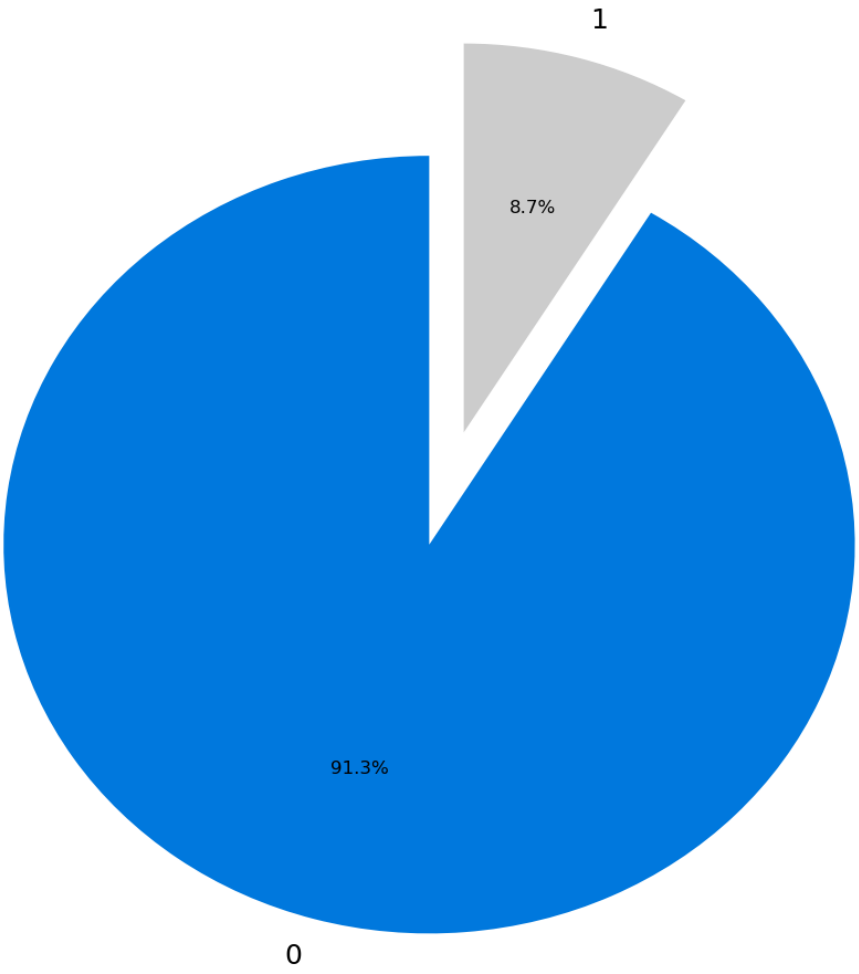


Figure B.4: Activity Duration in Seconds. The class mapping can be found in Chavarriaga et al. (2013).

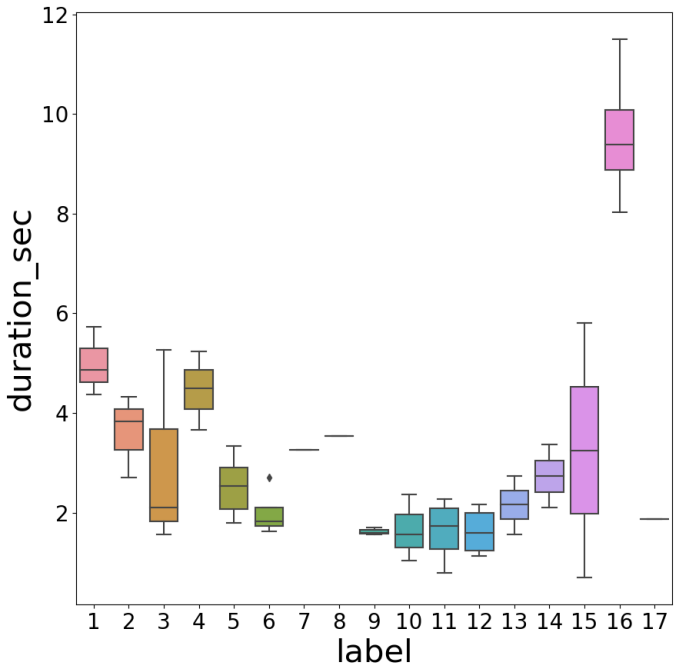


Figure B.6: Activity Duration in Seconds. The 'Freeze of Gait' activity is mapped as 1.

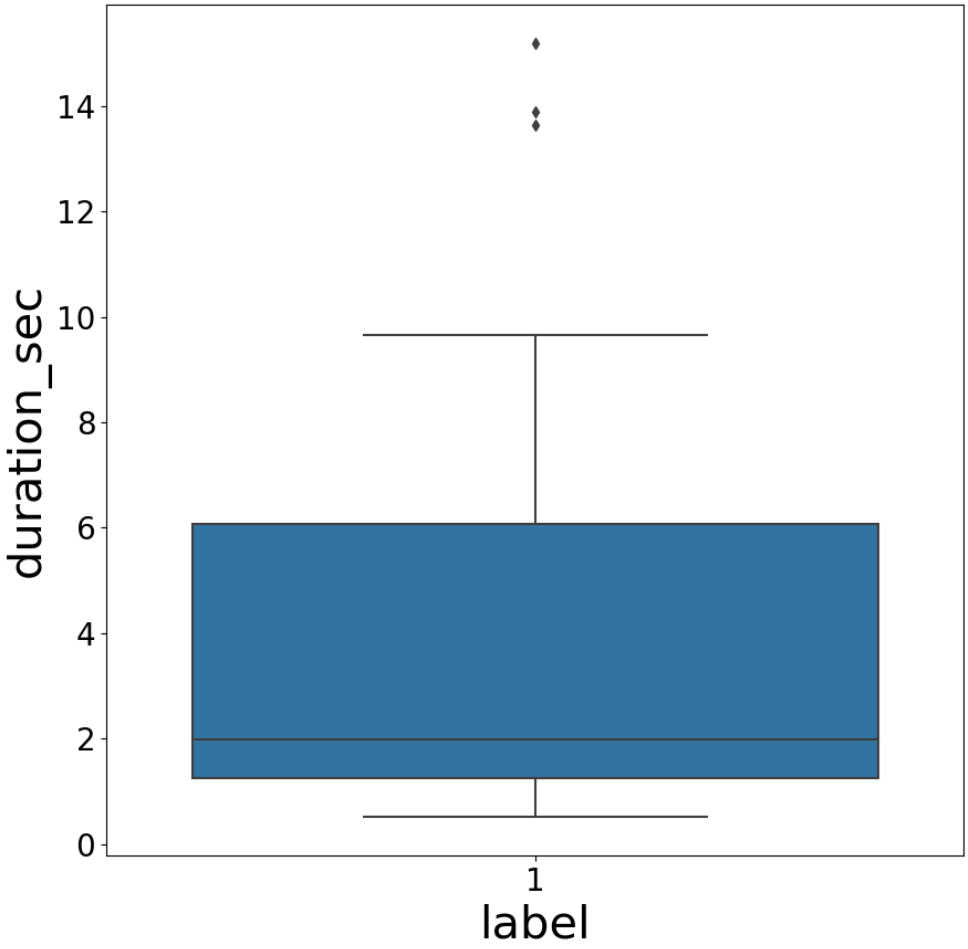


Table B.1: MMA Dataset Overview.

Match	Round	Subject A	Subject B
1	1	Fighter 1	Fighter 2
1	2	Fighter 1	Fighter 2
2	1	Fighter 3	Fighter 4
2	2	Fighter 3	Fighter 4
2	3	Fighter 3	Fighter 4

Figure B.7: Activity Distribution.

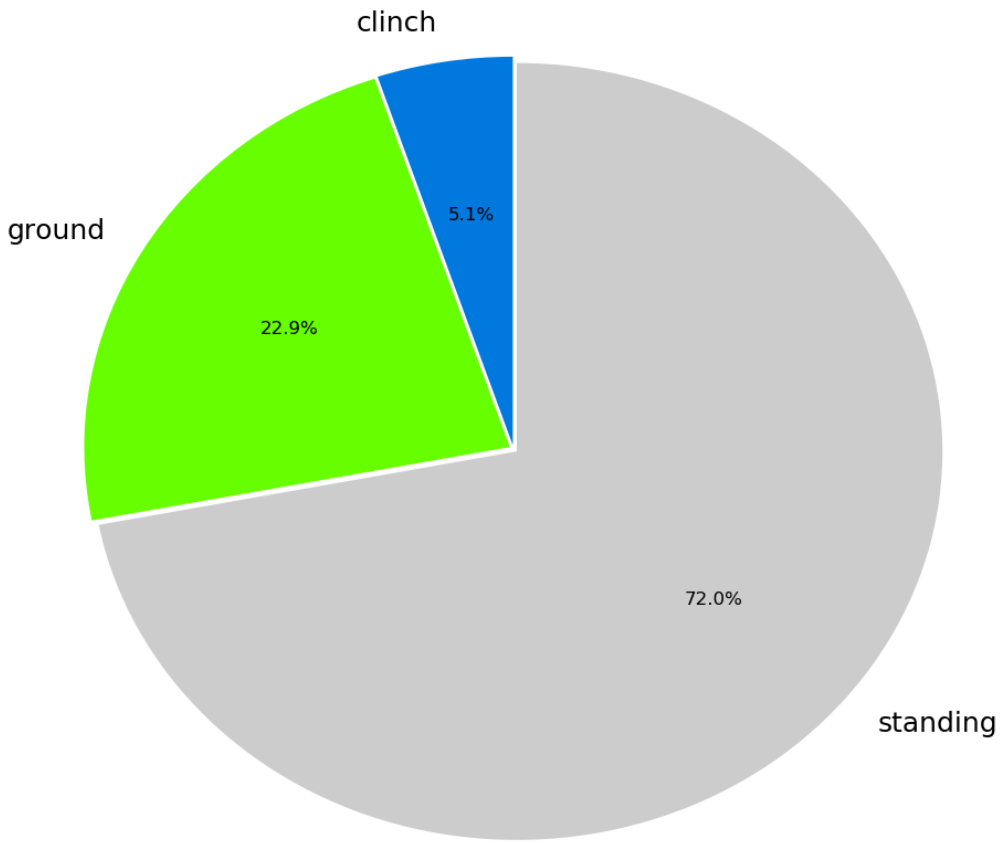
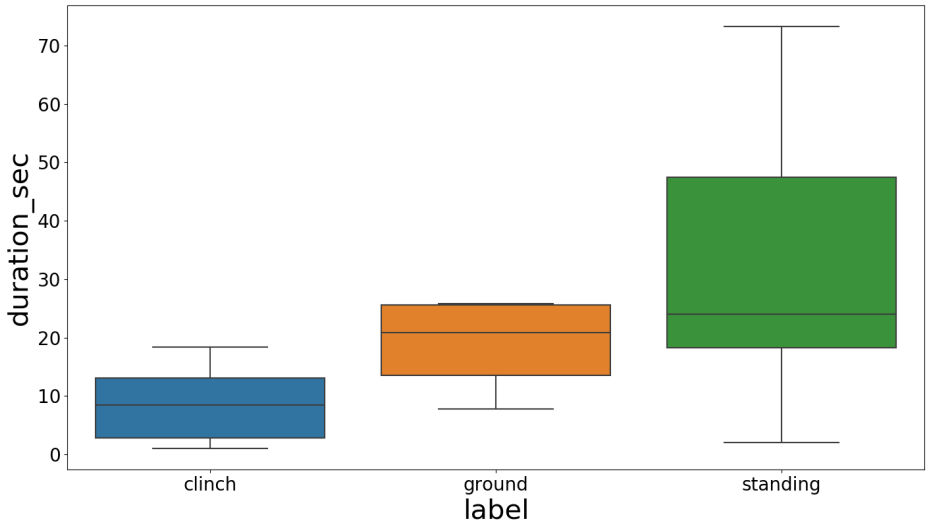


Figure B.8: Activity Duration in Seconds.



B.2 Experiment Results

B.2.1 REALDISP

Table B.2: REALDISP: Model Architectures.

RF				
Trees	**Feature Subset**	**Tree Depth**	**Criterion**	**Min. Sample Split**
600	$\sqrt{M}$	15	Gini	2
LSTM				
Layer	**Type**	**Units / Layer Specifics**	**Parameters**	**Dropout**
1	LSTM	786	2.528.256	$p = 0.5$
2	LSTM	512	2.623.488	$p = 0.5$
3	Output	34	2.611.234	-
$\sum$			**7.762.978**	
GRU				
Layer	**Type**	**Units / Layer Specifics**	**Parameters**	**Dropout**
1	GRU	128	70.272	$p = 0.2$
2	GRU	64	37.056	$p = 0.2$
3	Output	34	326.434	-
$\sum$			**433.762**	
CONV_LSTM				
Layer	**Type**	**Units / Layer Specifics**	**Parameters**	**Dropout**
1	CONV_1*D*	kernel:10,filter:64,stride:1	34.624	-
2	CONV_1*D*	kernel:10,filter:64,stride:1	41.024	-
3	MAXPOOL	pool_size:2,stride:2	0	-
4-5	CONV_1*D*	kernel:10,filter:64,stride:1	41.024	-
3	MAXPOOL	pool_size:2,stride:2	0	-
4	LSTM	128	98.816	$p = 0.2$
5	LSTM	64	49.408	$p = 0.2$
6	Output	34	80.546	-
$\sum$			**386.466**	
RES_LSTM				
Layer	**Type**	**Units / Layer Specifics**	**Parameters**	**Dropout**
1	LSTM	128	93.696	$p = 0.2$
2	LSTM	128	131.584	$p = 0.2$
3	ADD	$Layer1 + Layer2$	0	-
6	Output	~~34~~ 109	652.834	-
$\sum$			**878.114**	

B.2.2 OPPORTUNITY

Table B.3: OPPORTUNITY: Model Architectures.

RF				
Trees	**Feature Subset**	**Tree Depth**	**Criterion**	**Min. Sample Split**
300	All	20	Gini	2
LSTM				
Layer	**Type**	**Units / Layer Specifics**	**Parameters**	**Dropout**
1	LSTM	128	105.472	$p = 0.5$
2	LSTM	64	49.408	$p = 0.5$
3	Output	18	27.666	-
$\sum$			**182.546**	
GRU				
Layer	**Type**	**Units / Layer Specifics**	**Parameters**	**Dropout**
1	GRU	128	79.104	$p = 0.5$
2	GRU	64	37.056	$p = 0.5$
3	Output	18	27.666	-
$\sum$			**143.826**	
CONV_LSTM				
Layer	**Type**	**Units / Layer Specifics**	**Parameters**	**Dropout**
1	CONV_1*D*	kernel:2,filter:512,stride:1	79.360	-
2	CONV_1*D*	kernel:2,filter:512,stride:1	524.800	-
3	MAXPOOL	pool_size:2, stride:2	0	-
4	LSTM	128	328.192	$p = 0.5$
5	LSTM	64	49.408	$p = 0.5$
6	Output	18	13.842	-
$\sum$			**995.602**	
RES_LSTM				
Layer	**Type**	**Units / Layer Specifics**	**Parameters**	**Dropout**
1	LSTM	64	36.352	$p = 0.5$
2	LSTM	64	33.024	$p = 0.5$
3	ADD	$Layer1 + Layer2$	0	-
4	Output	18	27.666	-
$\sum$			**97.042**	

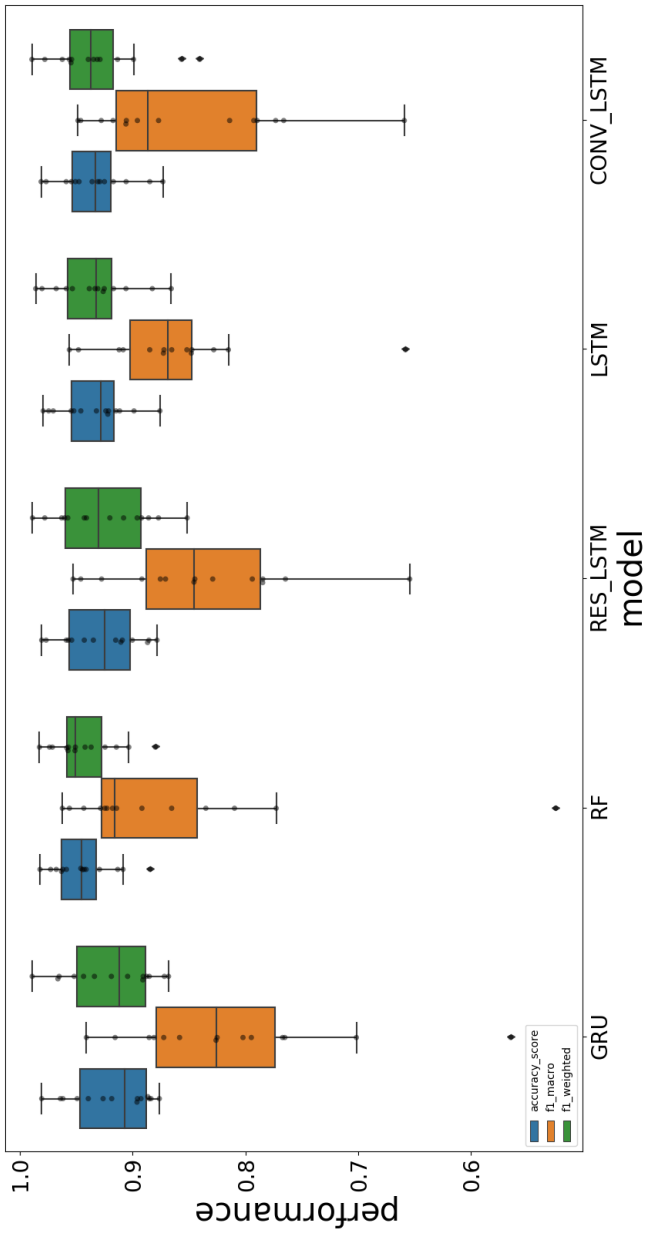


Figure B.9: LOSOCV Result for the Ideal Scenario of REALDISP.

Figure B.10: REALDISP: Random Forest Confusion Matrix for Train, Validation and Test Split.

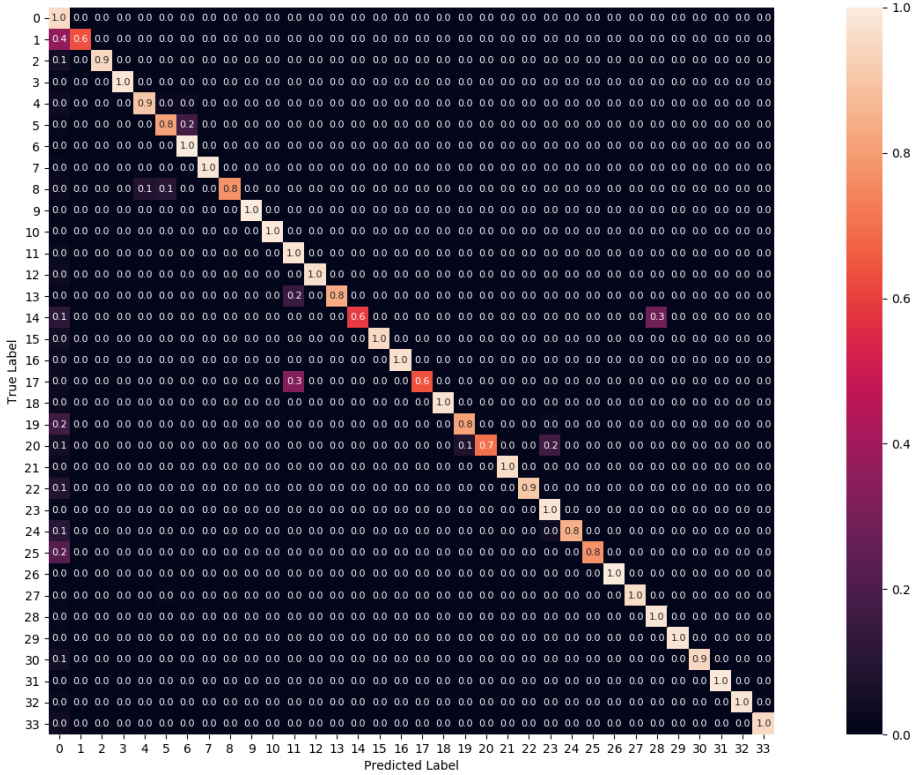


Figure B.11: REALDISP: LSTM Confusion Matrix for Train, Validation and Test Split.

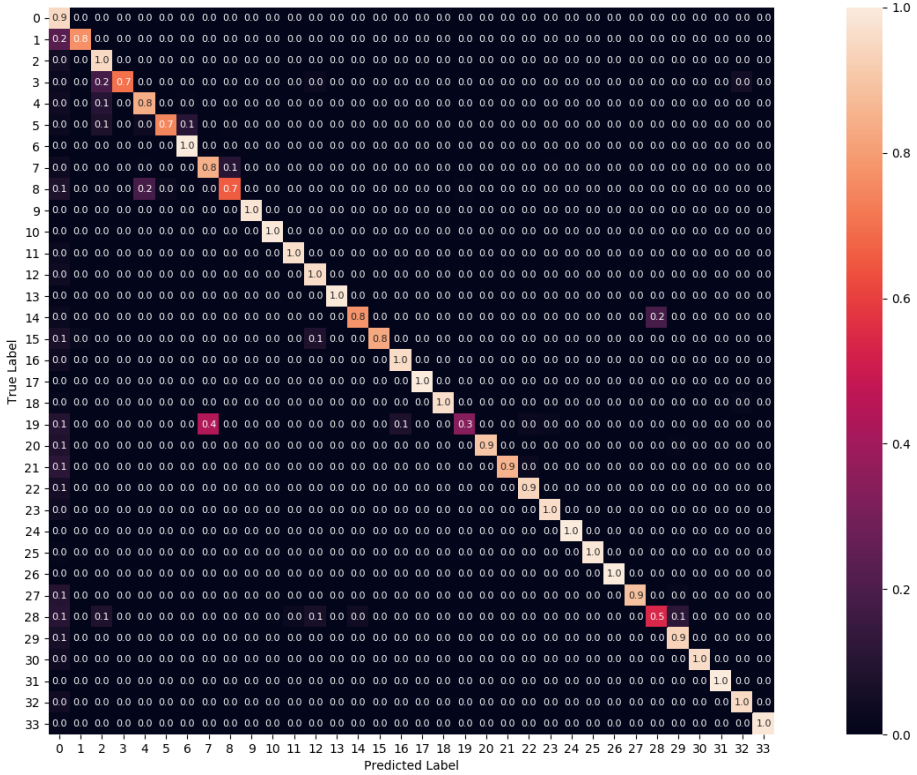


Figure B.12: REALDISP: GRU Confusion Matrix for Train, Validation and Test Split.

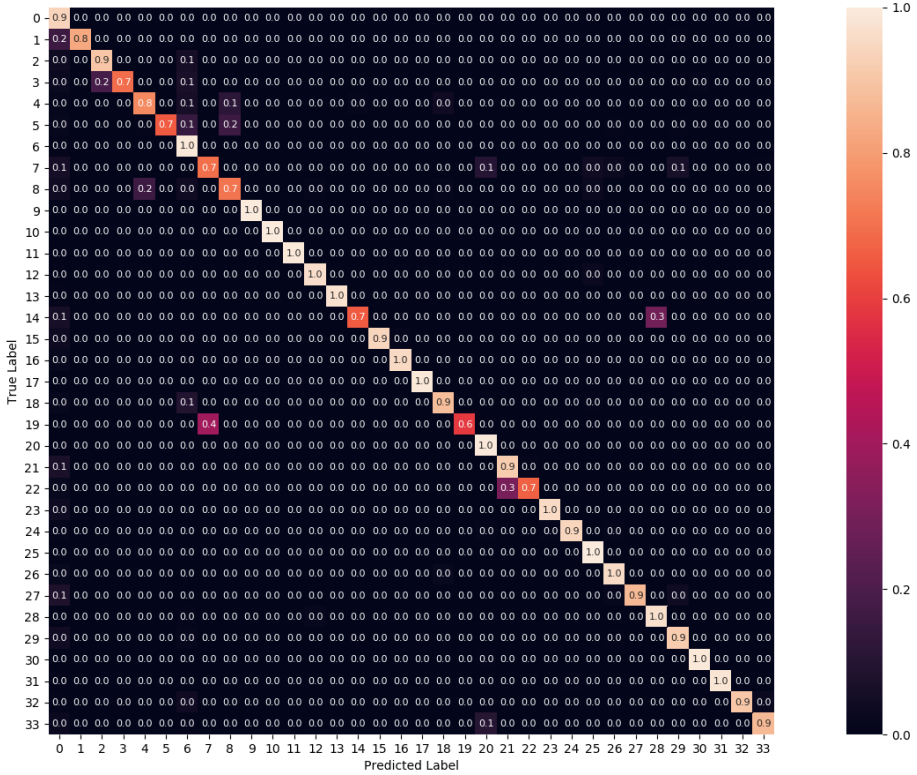


Figure B.13: REALDISP: CONV_LSTM Confusion Matrix for Train, Validation and Test Split.

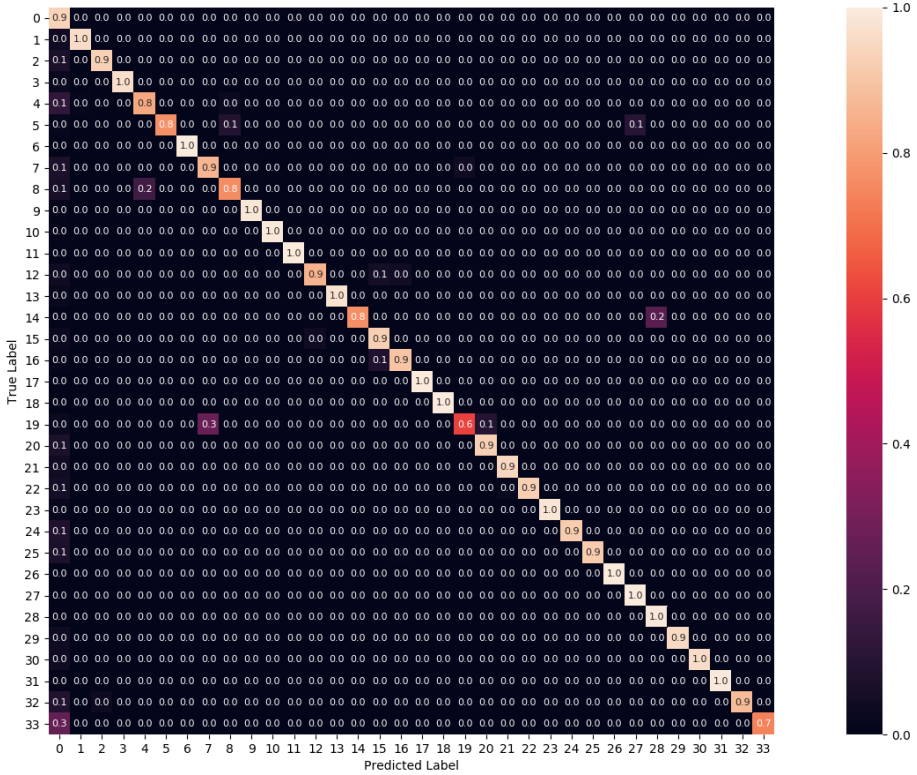
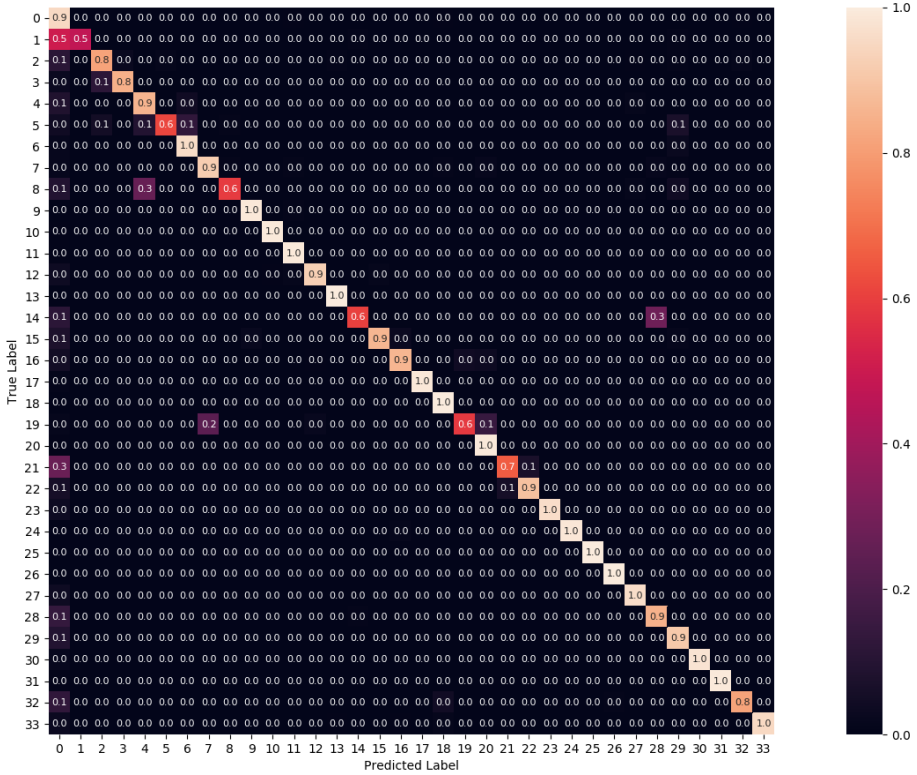


Figure B.14: REALDISP: RES_LSTM Confusion Matrix for Train, Validation and Test Split.



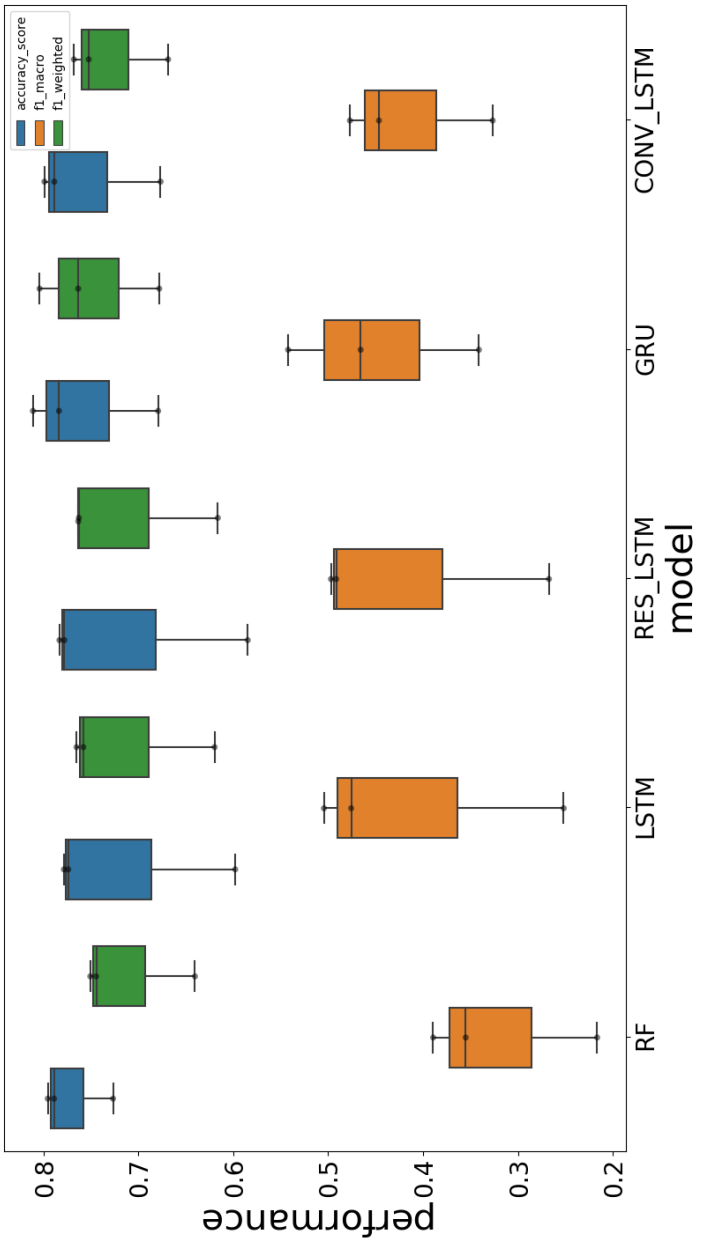


Figure B.15: LOSOCV Result for OPPORTUNITY.

Figure B.16: OPPORTUNITY: Random Forest Confusion Matrix for Train, Validation and Test Split.

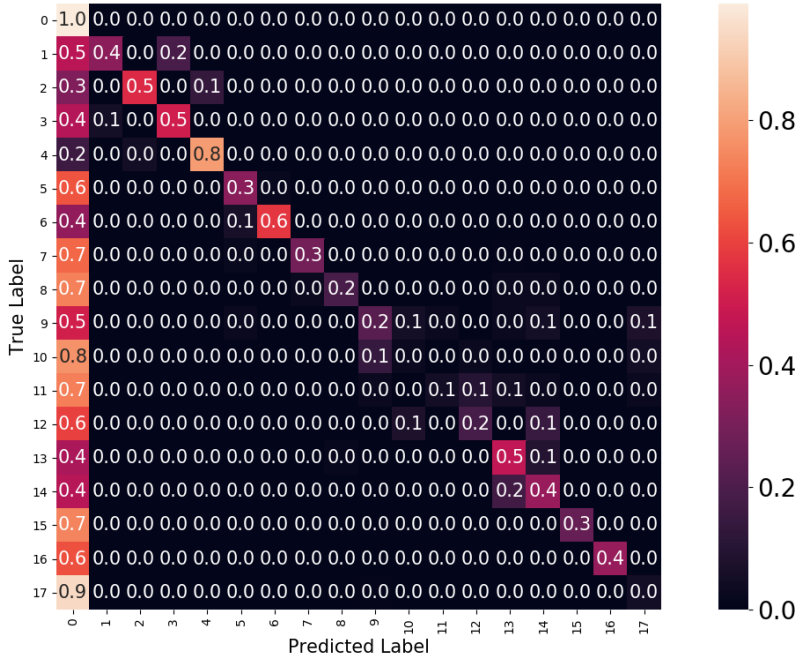


Figure B.17: OPPORTUNITY: LSTM Confusion Matrix for Train, Validation and Test Split.

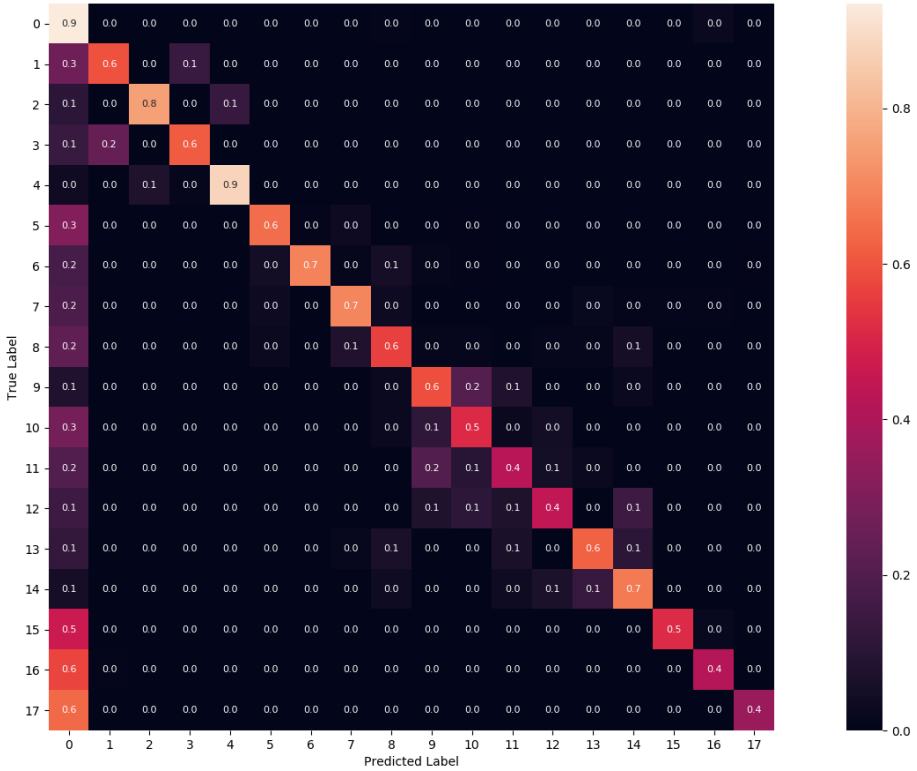


Figure B.18: OPPORTUNITY: GRU Confusion Matrix for Train, Validation and Test Split.

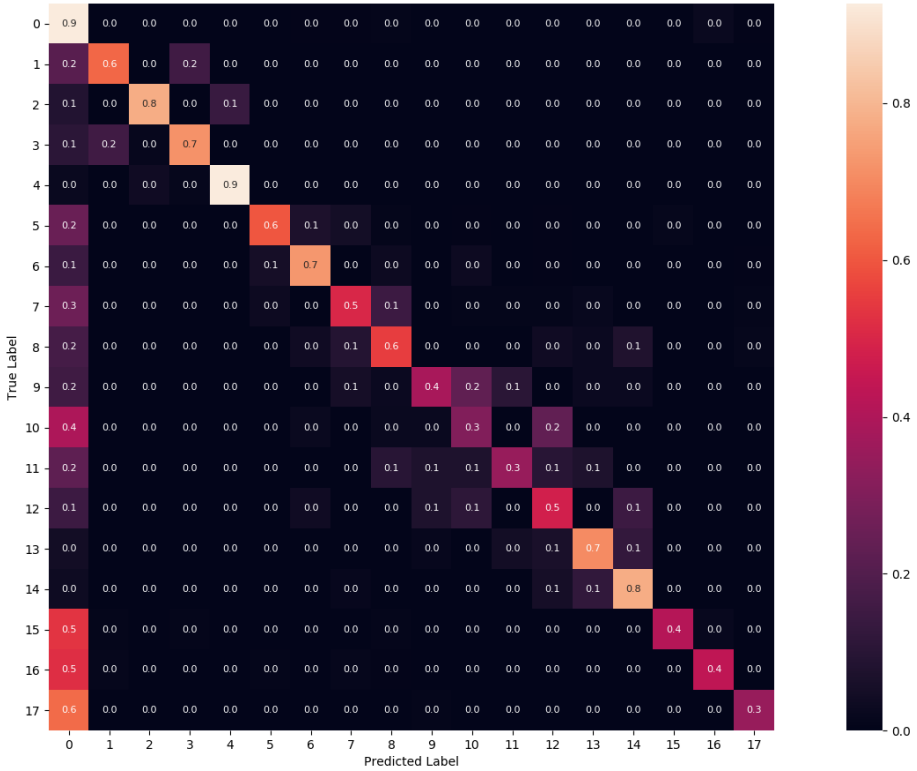


Figure B.19: OPPORTUNITY: CONV_LSTM Confusion Matrix for Train, Validation and Test Split.

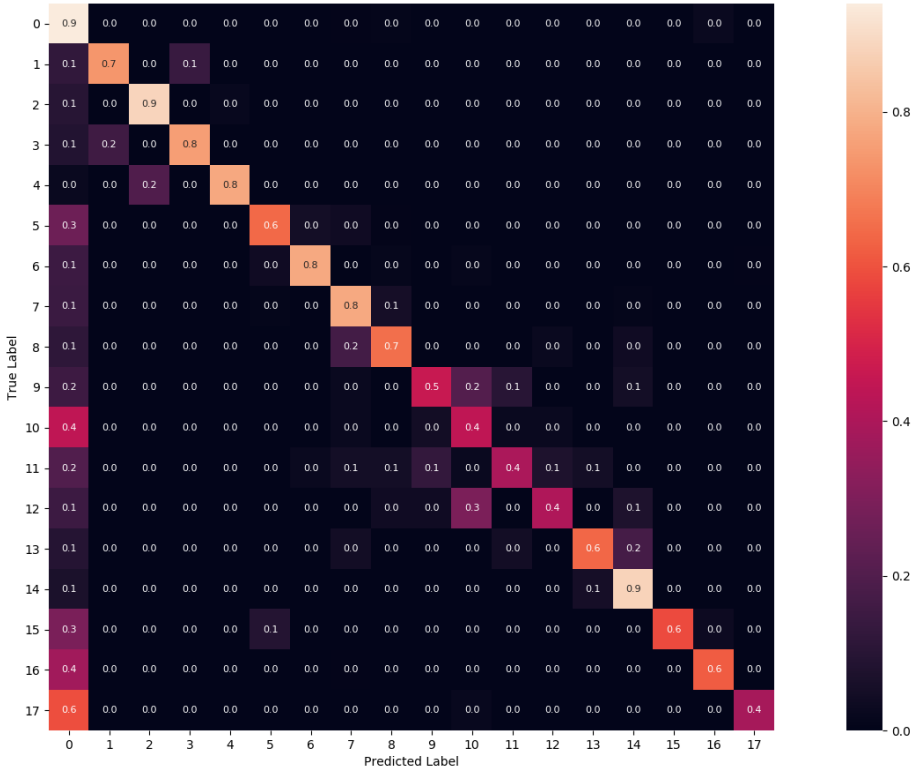
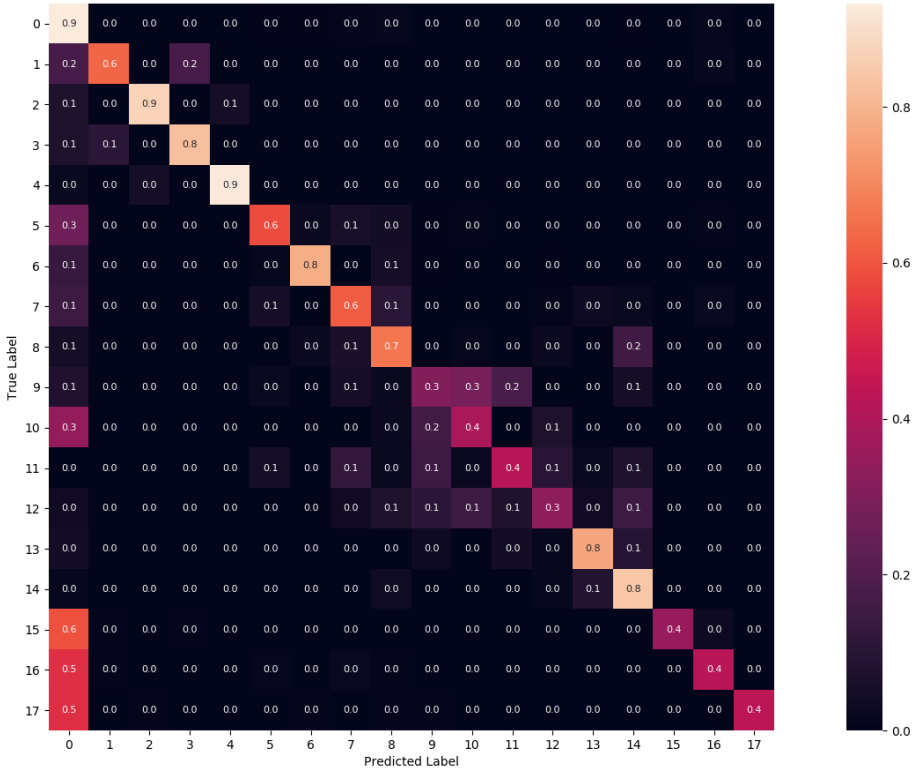


Figure B.20: OPPORTUNITY: RES_LSTM Confusion Matrix for Train, Validation and Test Split.



B.2.3 DAPHNET

Table B.4: DAPHNET: Model Architectures.

RF				
Trees	**Feature Subset**	**Tree Depth**	**Criterion**	**Min. Sample Split**
1000	*All*	5	Gini	2
LSTM				
Layer	**Type**	**Units / Layer Specifics**	**Parameters**	**Dropout**
1	LSTM	128	70.656	$p = 0.8$
2	LSTM	64	49.408	$p = 0.8$
3	LSTM	32	12.416	$p = 0.8$
4	Output	2	4.098	-
$\sum$			**136.578**	
GRU				
Layer	**Type**	**Units / Layer Specifics**	**Parameters**	**Dropout**
1	GRU	64	14.208	$p = 0.8$
2	GRU	64	24.768	$p = 0.8$
3	GRU	64	24.768	$p = 0.8$
4	Output	2	8.194	-
$\sum$			**71.938**	
CONV_LSTM				
Layer	**Type**	**Units / Layer Specifics**	**Parameters**	**Dropout**
1	CONV_1*D*	kernel:2,filter:32,stride:1	608	-
2	CONV_1*D*	kernel:2,filter:32,stride:1	2.080	-
3	MAXPOOL	pool_size:2, stride:2	0	-
4	LSTM	128	82.432	$p = 0.5$
5	LSTM	64	49.408	$p = 0.5$
6	Output	2	4.098	-
$\sum$			**138.626**	
RES_LSTM				
Layer	**Type**	**Units / Layer Specifics**	**Parameters**	**Dropout**
1	LSTM	32	5.376	$p = 0.5$
2	LSTM	32	8.320	$p = 0.5$
3	ADD	$Layer1 + Layer2$	0	-
4	LSTM	32	8.320	$p = 0.5$
5	ADD	$Layer4 + Layer3$	0	-
6	Output	2	4.098	-
$\sum$		124	**26.114**	

Figure B.21: LOSOCV Result for DAPHNET.

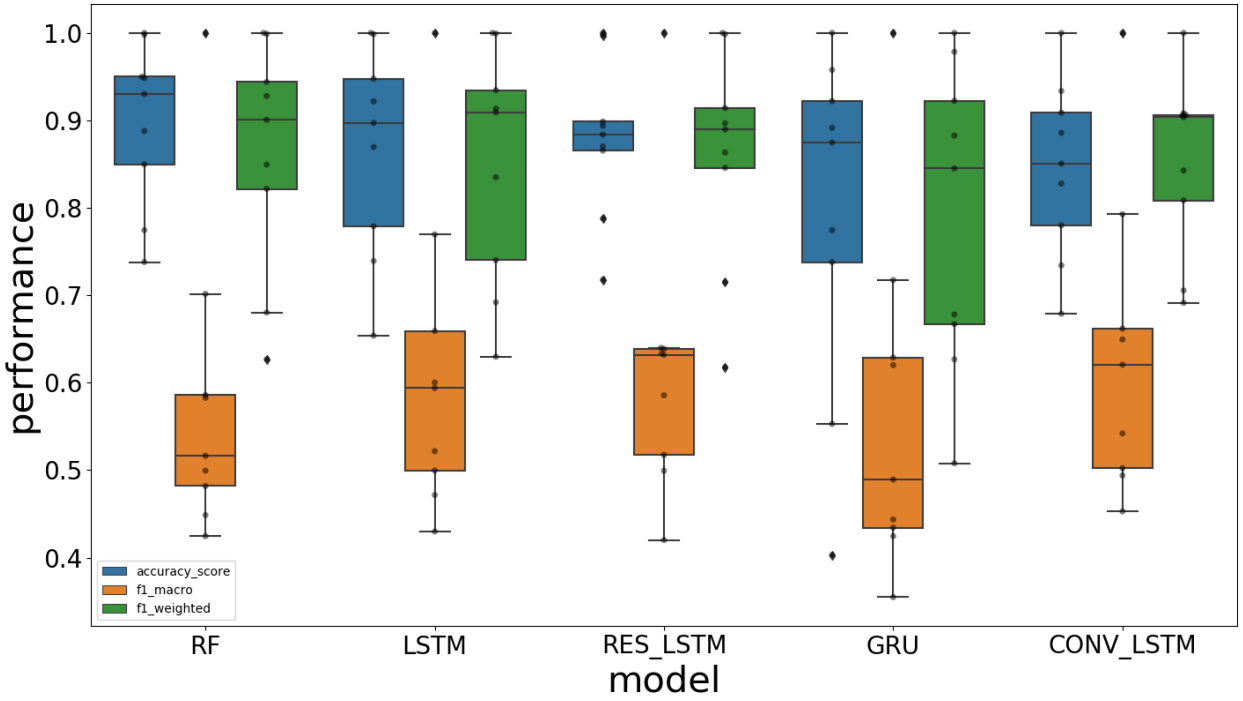


Figure B.22: DAPHNET: Random Forest Confusion Matrix for Train, Validation and Test Split.

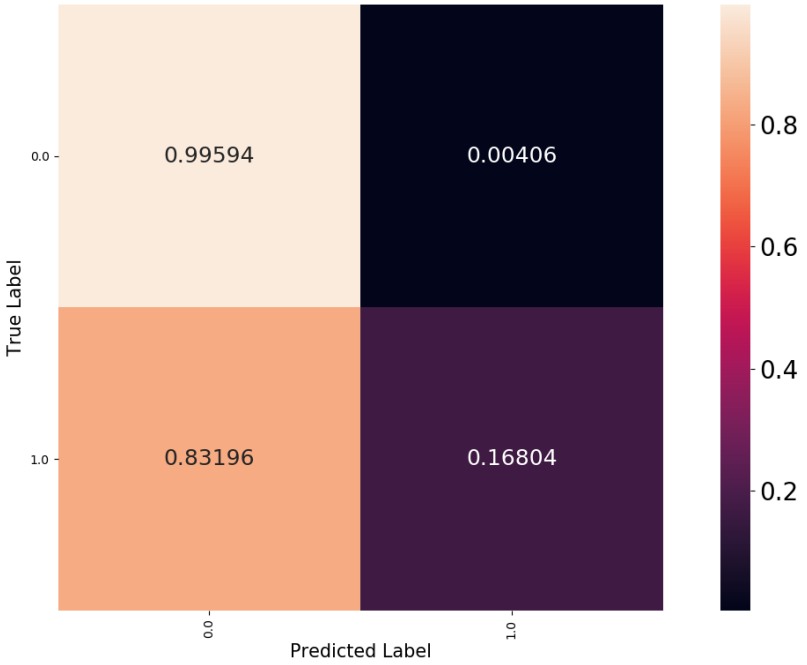


Figure B.23: DAPHNET: LSTM Confusion Matrix for Train, Validation and Test Split.

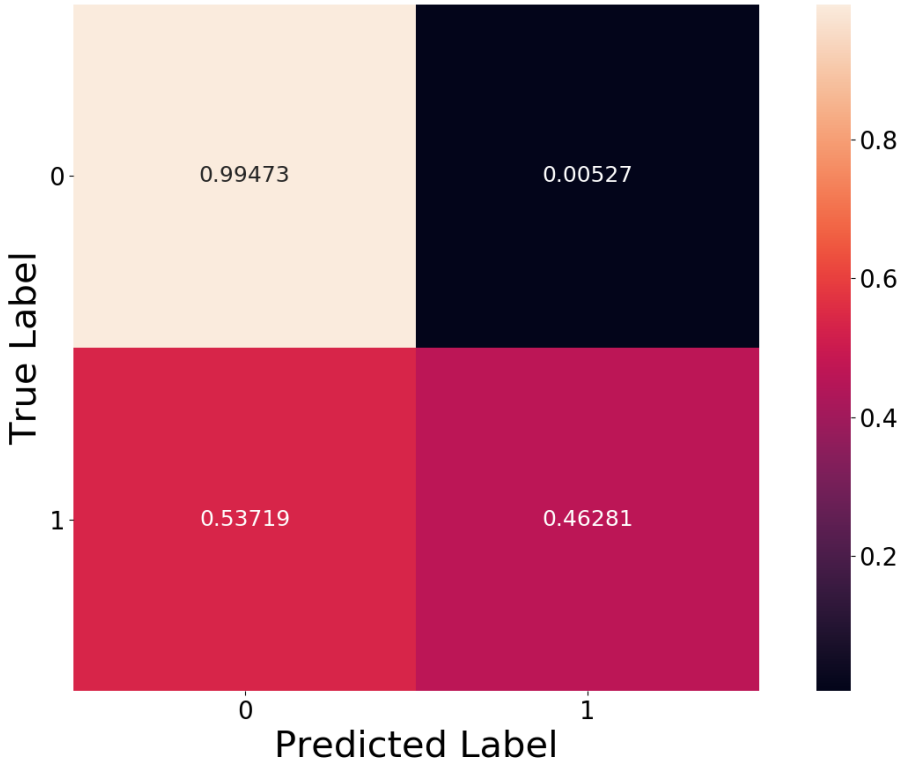


Figure B.24: DAPHNET: GRU Confusion Matrix for Train, Validation and Test Split.

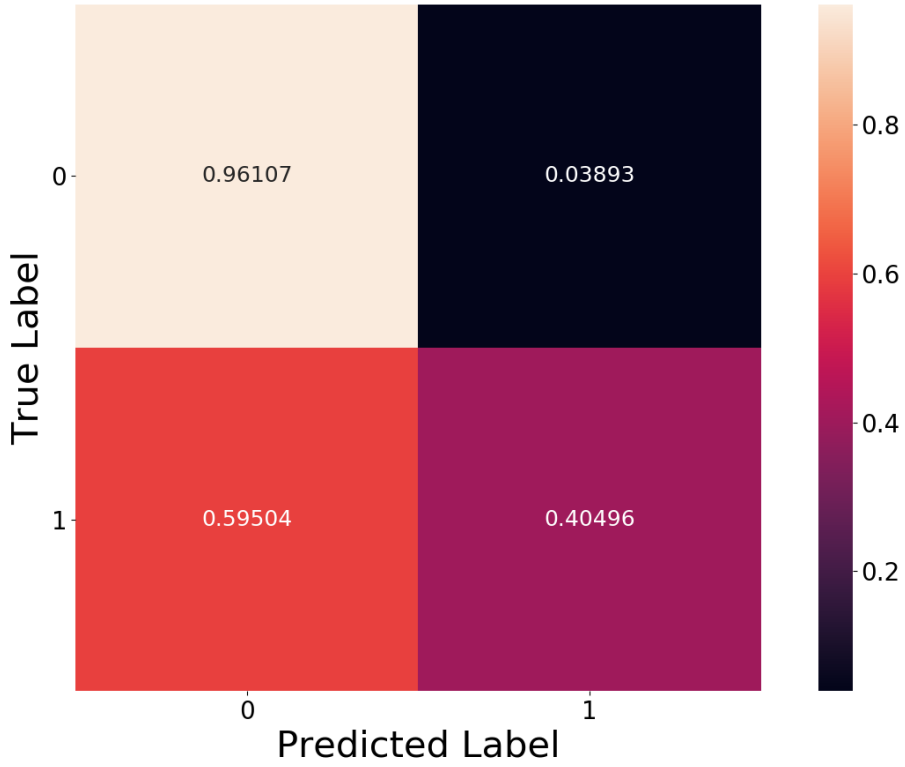


Figure B.25: DAPHNET: CONV_LSTM Confusion Matrix for Train, Validation and Test Split.

Figure B.26: DAPHNET: RES_LSTM Confusion Matrix for Train, Validation and Test Split.

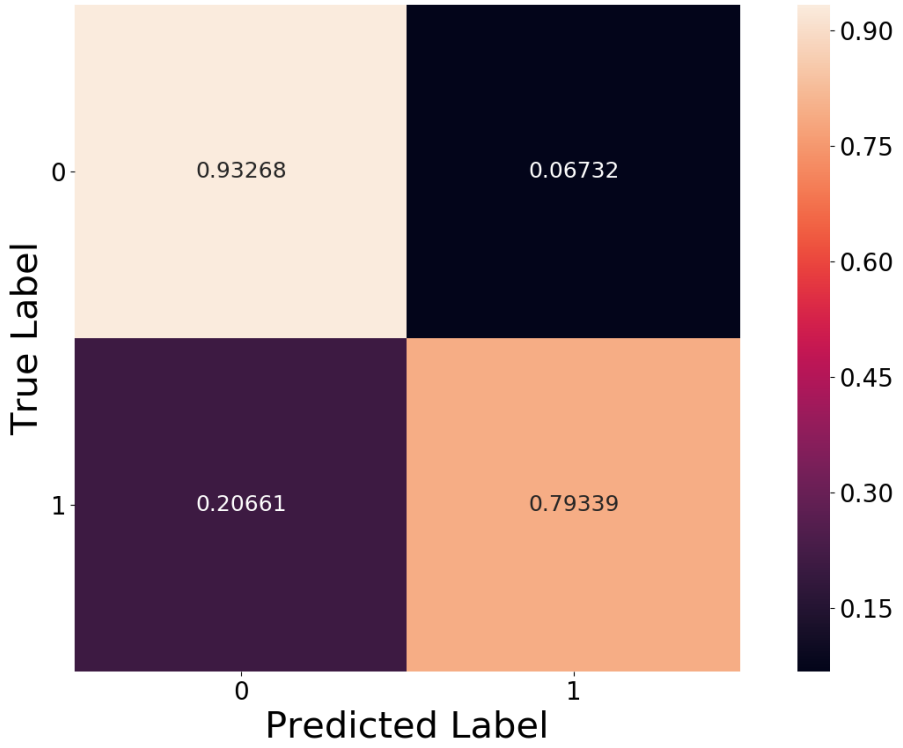
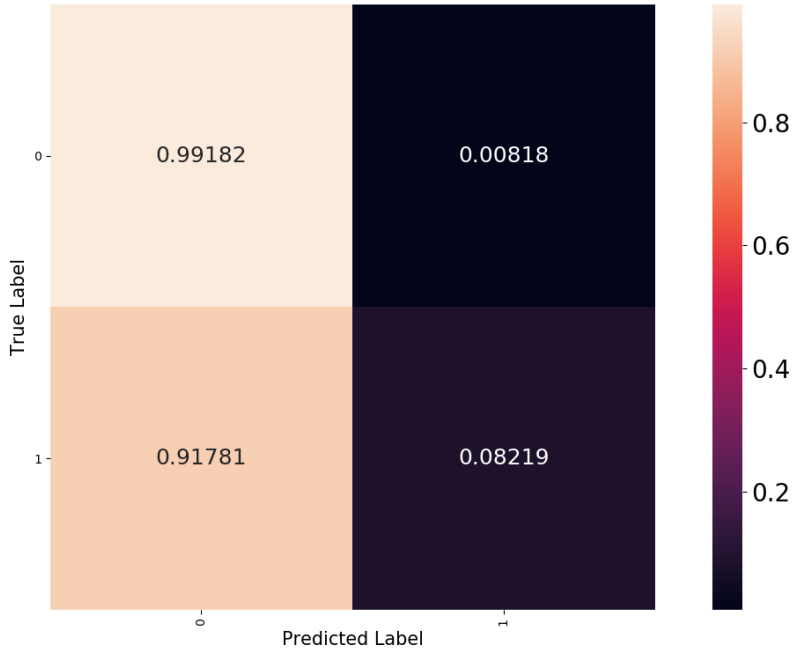


Figure B.27: DAPHNET: Random Forest LOSOCV Confusion Matrix. Label 1 is the Freeze of Gait Activity. The Confusion Matrix is first Averaged based on the LOSOCV Scores and then Normalized.



B.2.4 MMA

Table B.5: MMA: Model Architectures.

RF				
Trees	**Feature Subset**	**Tree Depth**	**Criterion**	**Min. Sample Split**
100	$\sqrt{M}$	15	Gini	2
LSTM				
Layer	**Type**	**Units / Layer Specifics**	**Parameters**	**Dropout**
1	LSTM	32	6.528	$p = 0.5$
2	LSTM	32	8.320	$p = 0.5$
3	LSTM	32	8.320	$p = 0.5$
4	Output	3	2.883	-
$\sum$			**26.051**	
GRU				
Layer	**Type**	**Units / Layer Specifics**	**Parameters**	**Dropout**
1	GRU	64	15.936	$p = 0.5$
2	GRU	32	9.312	$p = 0.5$
3	GRU	16	2.352	$p = 0.5$
4	Output	3	1.443	-
$\sum$			**29.043**	
CONV_LSTM				
Layer	**Type**	**Units / Layer Specifics**	**Parameters**	**Dropout**
1	CONV_1*D*	kernel:2,filter:512,stride:1	18.944	-
2	CONV_1*D*	kernel:2,filter:512,stride:1	524.800	-
3	MAXPOOL	pool_size:2, stride:2	0	-
4	LSTM	128	328.192	$p = 0.8$
5	LSTM	64	49.408	$p = 0.8$
6	Output	3	2.883	-
$\sum$			**924.227**	
RES_LSTM				
Layer	**Type**	**Units / Layer Specifics**	**Parameters**	**Dropout**
1	LSTM	128	75.264	$p = 0.5$
2	LSTM	128	131.584	$p = 0.5$
3	ADD	$Layer1 + Layer2$	0	-
4	Output	3	11.523	-
$\sum$			**218.371**	

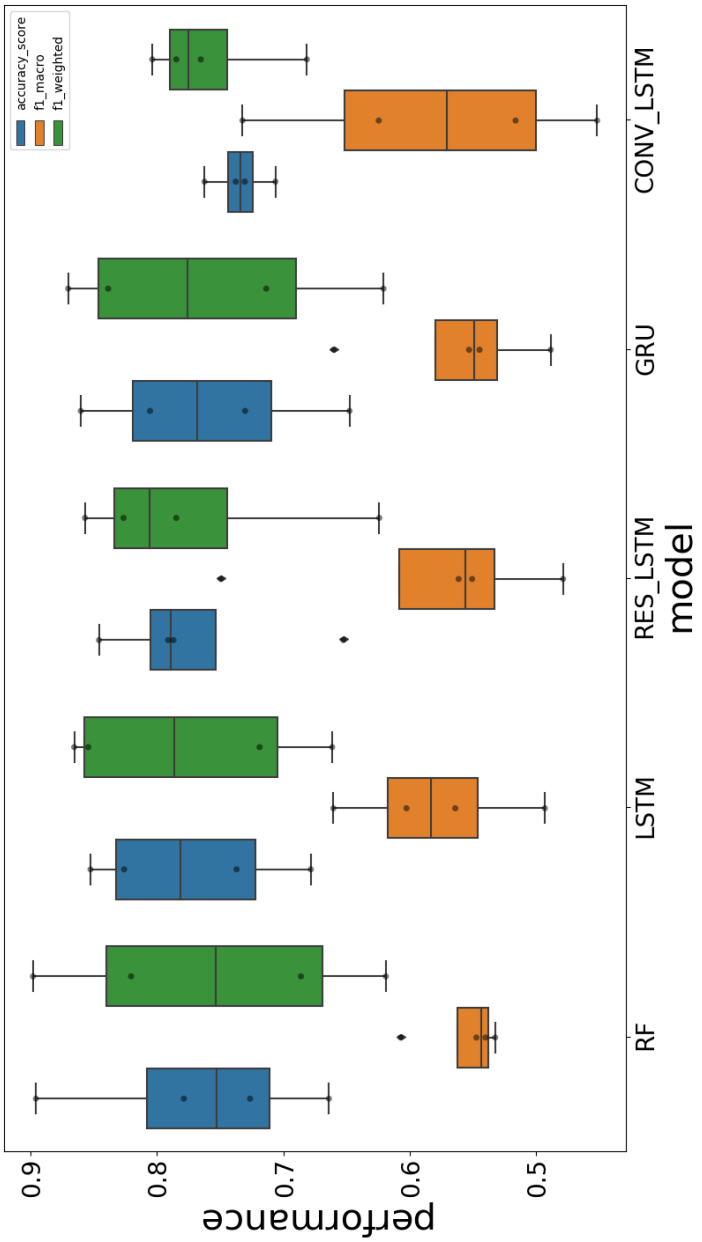


Figure B.28: LOSOCV Result for MMA.

Figure B.29: MMA: Random Forest Confusion Matrix for Train, Validation and Test Split.

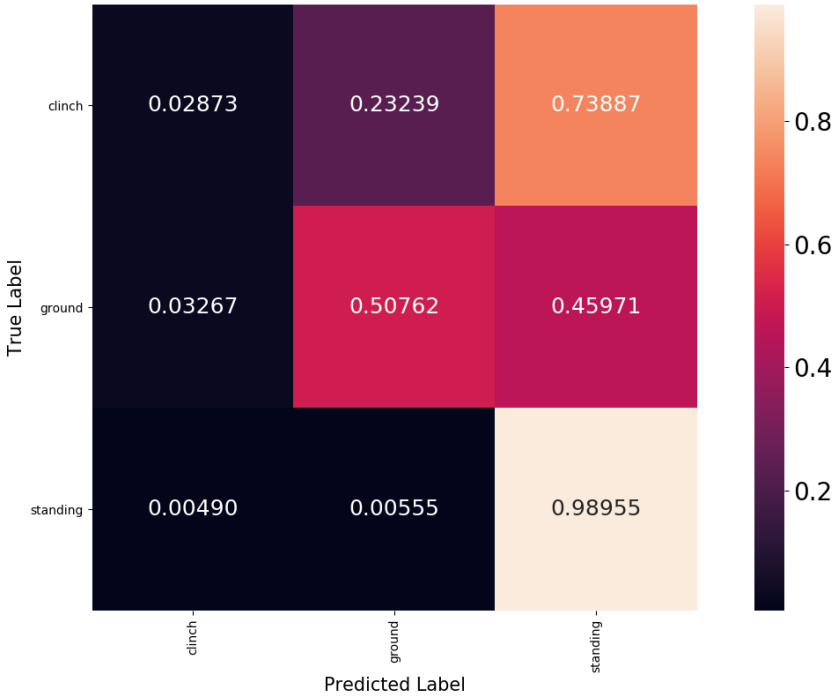


Figure B.30: MMA: LSTM Confusion Matrix for Train, Validation and Test Split.

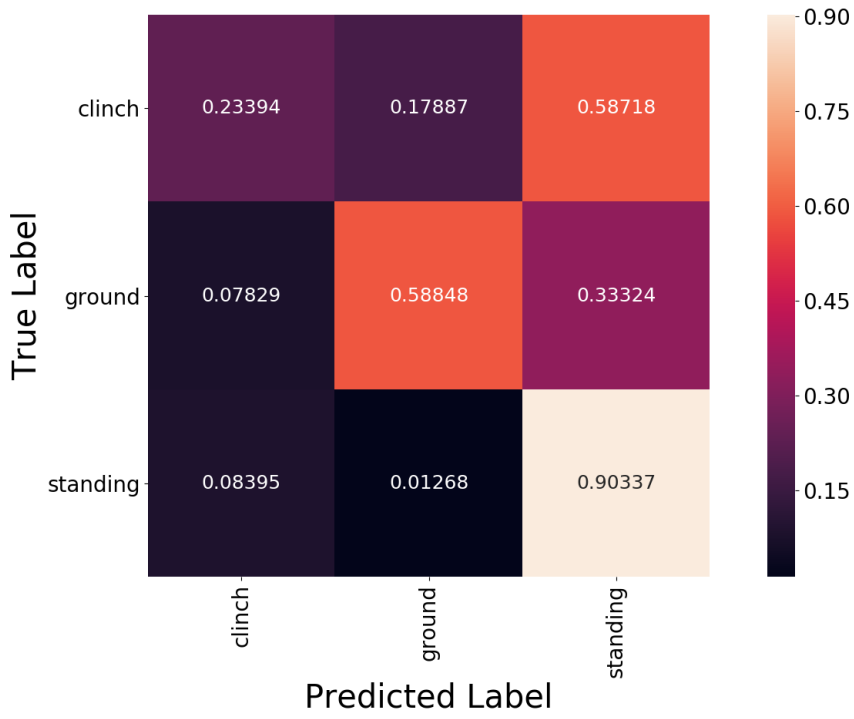


Figure B.31: MMA: GRU Confusion Matrix for Train, Validation and Test Split.

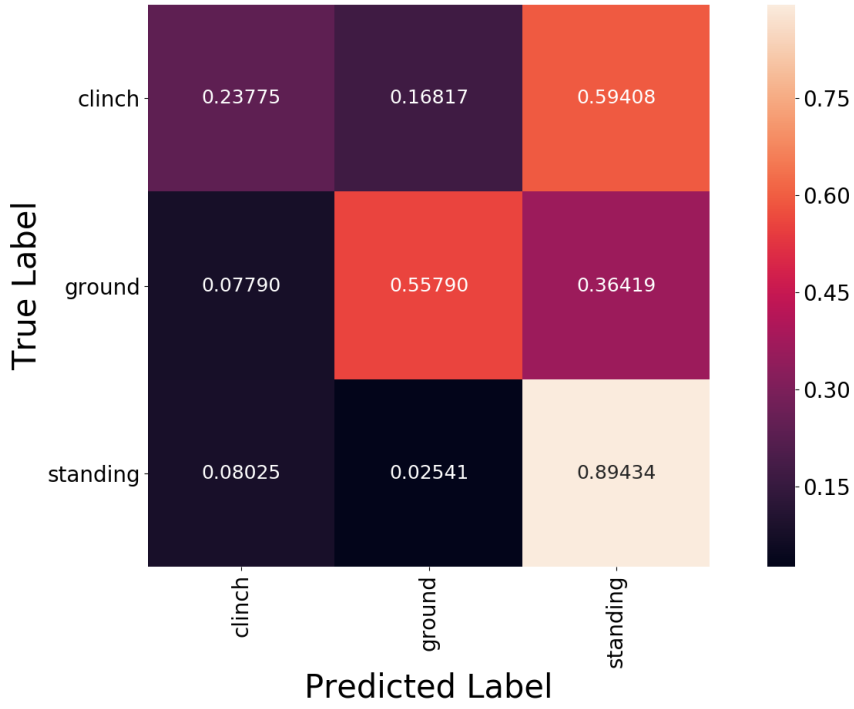


Figure B.32: MMA: CONV_LSTM Confusion Matrix for Train, Validation and Test Split.

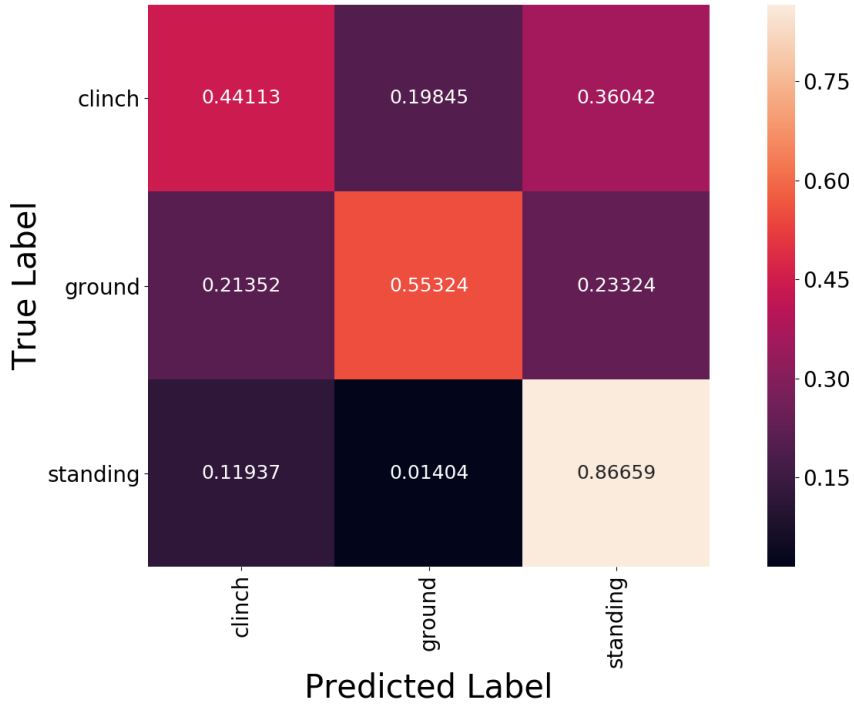
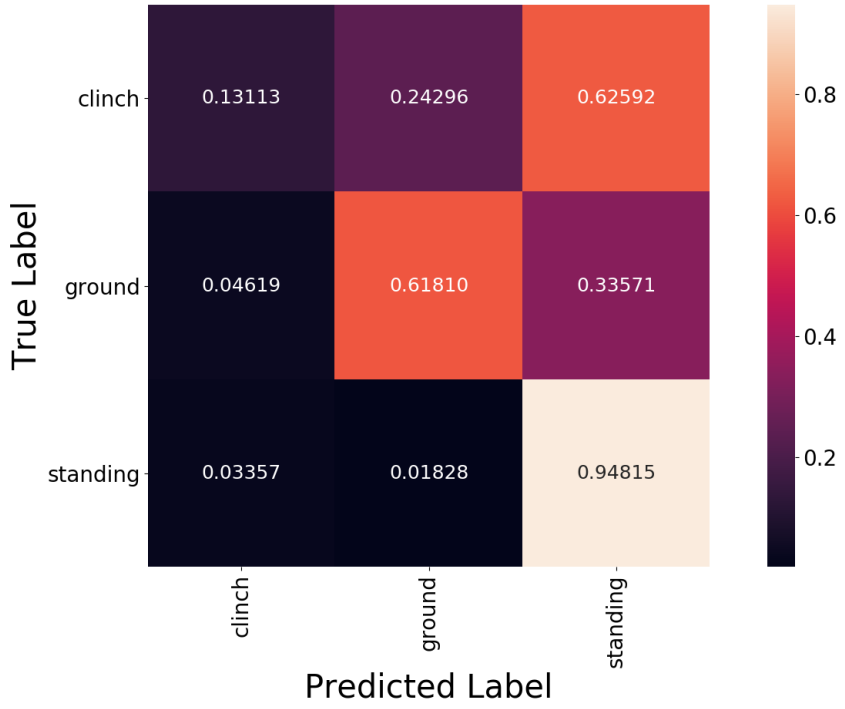


Figure B.33: MMA: RES_LSTM Confusion Matrix for Train, Validation and Test Split.



References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., & Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Albelwi, S. & Mahmood, A. (2017). A framework for designing the architectures of deep convolutional neural networks. *Entropy*, 19(6).

Autonomous Robots Lab, A. R. L. (2018). Autonomous robots lab. http://www.autonomousrobotslab.com/inertial-sensors.html. 2018-03-24.

Baños, O., Damas, M., Pomares, H., Rojas, I., Tóth, M. A., & Amft, O. (2012). A benchmark dataset to evaluate sensor displacement in activity recognition. In *Proceedings of the*

2012 ACM Conference on Ubiquitous Computing, UbiComp '12 (pp. 1026–1035). New York, NY, USA: ACM.

Baechlin, M., Roggen, D., Troester, G., Plotnik, M., Inbar, N., Maidan, I., Herman, T., Brozgol, M., Shaviv, E., Giladi, N., & Hausdorff, J. M. (2009). Potentials of enhanced context awareness in wearable assistants for parkinson's disease patients with the freezing of gait syndrome. In *ISWC* (pp. 123–130).: IEEE Computer Society.

Baldominos, A., Isasi, P., & Saez, Y. (2017). Feature selection for physical activity recognition using genetic algorithms. In *2017 IEEE Congress on Evolutionary Computation (CEC)* (pp. 2185–2192).

Bao, L. & Intille, S. S. (2004). Activity recognition from user-annotated acceleration data. In A. Ferscha & F. Mattern (Eds.), *Pervasive Computing* (pp. 1–17). Berlin, Heidelberg: Springer Berlin Heidelberg.

Bartalesi, R., Lorussi, F., Tesconi, M., Tognetti, A., Zupone, G., & De Rossi, D. (2005). Wearable kinesthetic system for capturing and classifying upper limb gesture. In *Eurohaptics Conference, 2005 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2005. World Haptics 2005. First Joint* (pp. 535–536).: IEEE.

Bayer, J. S. (2015). *Learning Sequence Representations*. PhD thesis, München.

Bishop, C. M. (2006). *Deep Learning*. Springer-Verlag New York.

Blanke, U. & Schiele, B. (2009). Daily routine recognition through activity spotting. In T. Choudhury, A. Quigley, T. Strang, & K. Suginuma (Eds.), *Location and Context Awareness* (pp. 192–206). Berlin, Heidelberg: Springer Berlin Heidelberg.

Blanke, U. & Schiele, B. (2010). Remember and transfer what you have learned-recognizing composite activities based on activity spotting. In *Wearable Computers (ISWC), 2010 International Symposium on* (pp. 1–8).: IEEE.

Bonato, P. (2003). Wearable sensors/systems and their impact on biomedical engineering. *IEEE Eng Med Biol Mag*, 22(3), 18–20.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5–32.

Bulling, A., Blanke, U., & Schiele, B. (2014). A tutorial on human activity recognition using body-worn inertial sensors. *ACM Comput. Surv.*, 46(3), 33:1–33:33.

Bulling, A. & Roggen, D. (2011). Recognition of visual memory recall processes using eye movement analysis. In *Proceedings of the 13th International Conference on Ubiquitous Computing*, UbiComp '11 (pp. 455–464). New York, NY, USA: ACM.

Chavarriaga, R., Sagha, H., Calatroni, A., Digumarti, S. T., Tröster, G., Millán, J. D. R., & Roggen, D. (2013). The opportunity challenge: A benchmark database for on-body sensor-based activity recognition. *Pattern Recogn. Lett.*, 34(15), 2033–2042.

Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *EMNLP 2014*.

Chollet, F. et al. (2015). Keras. https://github.com/keras-team/keras.

Chowdhury, A., Tjondronegoro, D., Chandran, V., & Trost, S. (2018). Physical activity recognition using posterior-adapted class-based fusion of multi-accelerometers data. *IEEE Journal of Biomedical and Health Informatics*, PP(99), 1–1.

Coillot, C. & Leroy, P. (2012). Induction magnetometers principle, modeling and ways of improvement. In *Magnetic Sensors-Principles and Applications*. InTech.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control*, 2, 303–314.

Duchi, J. C., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12, 2121–2159.

Fawcett, T. (2006). An introduction to roc analysis. *Pattern Recogn. Lett.*, 27(8), 861–874.

Glorot, X. & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh & M. Titterington (Eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research* (pp. 249–256). Chia Laguna Resort, Sardinia, Italy: PMLR.

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. In G. J. Gordon, D. B. Dunson, & M. Dudík (Eds.), *AISTATS*, volume 15 of *JMLR Proceedings* (pp. 315–323).: JMLR.org.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Graves, A. (2008). *Supervised sequence labelling with recurrent neural networks*. PhD thesis, Technical University Munich.

Graves, A., Mohamed, A.-r., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. *Acoustics, speech and signal processing (icassp), 2013 IEEE international conference on*, (pp. 6645–6649).

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2017). Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10), 2222–2232.

Hammerla, N. Y., Halloran, S., & Plötz, T. (2016). Deep, convolutional, and recurrent models for human activity recognition using wearables. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16 (pp. 1533–1540).: AAAI Press.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proceedings of the IEEE international conference on computer vision*, (pp. 1026–1034).

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (pp. 770–778).

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, *6*(2), 107–116.

Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, *9*(8), 1735–1780.

Huynh, T., Fritz, M., & Schiele, B. (2008). Discovery of activity patterns using topic models. In *Proceedings of the 10th International Conference on Ubiquitous Computing*, UbiComp '08 (pp. 10–19). New York, NY, USA: ACM.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2014). *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated.

Jiang, S., Cao, Y., Iyengar, S., Kuryloski, P., Jafari, R., Xue, Y., Bajcsy, R., & Wicker, S. (2008). Carenet: An integrated wireless sensor networking environment for remote healthcare. In *Proceedings of the ICST 3rd International Conference on Body Area Networks*, BodyNets '08 (pp. 9:1–9:3). ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15 (pp. 2342–2350).: JMLR.org.

Kaufman, S., Rosset, S., & Perlich, C. (2011). Leakage in data mining: Formulation, detection, and avoidance. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11 (pp. 556–563). New York, NY, USA: ACM.

Kautz, H., Arnstein, L., Borriello, G., Etzioni, O., & Fox, D. (2002). An overview of the assisted cognition project. In *AAAI-2002 Workshop on Automation as Caregiver: The Role of Intelligent Technology in Elder Care*, number 2002 (pp. 6065).

Kingma, D. & Ba, J. (2014). Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Kohavi, R. & John, G. H. (1997). Wrappers for feature subset selection. *Artif. Intell.*, 97(1-2), 273–324.

Kok, M., Hol, J. D., & Schön, T. B. (2017). Using inertial sensors for position and orientation estimation. *Found. Trends Signal Process.*, 11(1-2), 1–153.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Wein-

berger (Eds.), *Advances in Neural Information Processing Systems 25* (pp. 1097–1105). Curran Associates, Inc.

Lara, O. D. & Labrador, M. A. (2013). A Survey on Human Activity Recognition using Wearable Sensors. *IEEE Communications Surveys; Tutorials*, 15(3), 1192–1209.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4), 541–551.

LeCun, Y., Bottou, L., Orr, G. B., & Müller, K.-R. (1998). Efficient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop* (pp. 9–50). London, UK, UK: Springer-Verlag.

Li, F., Shirahama, K., Nisar, M. A., Köping, L., & Grzegorzek, M. (2018). Comparison of feature learning methods for human activity recognition using wearable sensors. *Sensors*, 18(2).

Light, W. (1992). Ridge functions, sigmoidal functions and neural networks. *Approximation theory VII*, (pp. 163–206).

Lockhart, J. W., Pulickal, T., & Weiss, G. M. (2012). Applications of mobile activity recognition. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12 (pp. 1054–1058). New York, NY, USA: ACM.

Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*.

Marsland, S. (2009). *Machine Learning: An Algorithmic Perspective*. Chapman & Hall/CRC, 1st edition.

Nike Inc., N. I. (2018). Official website of nike+. http://www.nikeplus.com. 2018-03-24.

Ordóñez, F. J. & Roggen, D. (2016). Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16(1).

Osmani, V., Balasubramaniam, S., & Botvich, D. (2007). Self-organising object networks using context zones for distributed activity recognition. In *Proceedings of the ICST 2Nd International Conference on Body Area Networks*, BodyNets '07 (pp. 14:1–14:9). ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Papagiannakis, G., Singh, G., & Magnenat-Thalmann, N. (2008). A survey of mobile and wireless technologies for augmented reality systems. *Computer Animation and Virtual Worlds*, 19(1), 3–22.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning* (pp. 1310–1318).

Patterson, D. W. (1996). *Artificial Neural Networks: Theory and Applications*. Prentice-Hall Series in Advanced Communications. Prentice Hall.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

Peng, H., Long, F., & Ding, C. (2005). Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on pattern analysis and machine intelligence*, 27(8), 1226–1238.

Polar Electro, P. E. E. G. (2018). Official website of polar. http://www.polar.com. 2018-03-24.

Proakis, J. G. & Manolakis, D. K. (2006). *Digital Signal Processing (4th Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Ravi, N., Dandekar, N., Mysore, P., & Littman, M. L. (2005). Activity recognition from accelerometer data. In *Proceedings of the 17th Conference on Innovative Applications of Artificial Intelligence - Volume 3*, IAAI'05 (pp. 1541–1546).: AAAI Press.

Reed, R. & Marks, R. (1999). *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. A Bradford book. MIT Press.

Ripley, B. D. (2007). *Pattern recognition and neural networks*. Cambridge university press.

San, P. P., Kakar, P., Li, X.-L., Krishnaswamy, S., Yang, J.-B., & Nguyen, M. N. (2017). Deep learning for human activity recognition. In *Big Data Analytics for Sensor-Network Collected Intelligence* (pp. 186–204). Elsevier.

Schlömer, T., Poppinga, B., Henze, N., & Boll, S. (2008). Gesture recognition with a wii controller. In *Proceedings of the 2Nd International Conference on Tangible and Embedded Interaction*, TEI '08 (pp. 11–14). New York, NY, USA: ACM.

Seide, F. & Agarwal, A. (2016). Cntk: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16 (pp. 2135–2135). New York, NY, USA: ACM.

Somol, P., Novovičová, J., & Pudil, P. (2006). Flexible-hybrid sequential floating search in statistical feature selection. In D.-Y. Yeung, J. T. Kwok, A. Fred, F. Roli, & D. de Ridder (Eds.), *Structural, Syntactic, and Statistical Pattern Recognition* (pp. 632–639). Berlin, Heidelberg: Springer Berlin Heidelberg.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15, 1929–1958.

Sun, X. & Cheney, E. (1992). The fundamentality of sets of ridge functions. *aequationes mathematicae*, 44(2-3), 226–235.

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13 (pp. III–1139–III–1147).: JMLR.org.

Tieleman, T. & Hinton, G. (2012). Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

Wassermann, L. (2006). All of nonparametric statistics. *New York*.

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zhang, J. & Zong, C. (2015). Deep neural networks in machine translation: An overview. *IEEE Intelligent Systems*, 30(5), 16–25.

Zhang, M. & Sawchuk, A. A. (2012). Motion primitive-based human activity recognition using a bag-of-features approach. In *Proceedings of the 2Nd ACM SIGHIT International Health Informatics Symposium*, IHI '12 (pp. 631–640). New York, NY, USA: ACM.

Zhou, Y. T. & Chellappa, R. (1988). Computation of optical flow using a neural network. In *IEEE 1988 International Conference on Neural Networks* (pp. 71–78 vol.2).

Zhu, J., San-Segundo, R., & Pardo, J. M. (2017). Feature extraction for robust physical activity recognition. *Human-centric Computing and Information Sciences*, 7(1), 16.

Zinnen, A., Wojek, C., & Schiele, B. (2009). Multi activity recognition based on bodymodel-derived primitives. In T. Choudhury, A. Quigley, T. Strang, & K. Suginuma (Eds.), *Location and Context Awareness* (pp. 1–18). Berlin, Heidelberg: Springer Berlin Heidelberg.

Declaration of Authorship

Erklärung

Ich erkläre, dass ich meine Masterarbeit ”Wearable Sensor based Human Activity Recognition with Recurrent Neural Networks” selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe und dass ich alle Stellen, die ich wörtlich oder sinngemäß aus Veröffentlichungen etnommen habe, als solche kenntlich gemacht habe. Die Arbeit hat bisher in gleicher oder ähnlicher Form oder auszugsweise noch keiner Prüfungsbehörde vorgelegen.
Ich versichere, dass die eingereichte schriftliche Fassung der auf dem beigefügten Medium gespeicherten Fassung entspricht.

Datum, Unterschrift