Objectively recognizing human activity in body-worn sensor data with (more or less) deep neural networks

SOFIA BROOMÉ

Master in Computer Science
Date: May 29, 2017
Supervisor: Josephine Sullivan
Examiner: Hedvig Kjellström
Swedish title: Objektiv igenkänning av mänsklig aktivitet från accelerometerdata med mer eller mindre djupa neurala nätverk
School of Computer Science and Communication

Abstract

This thesis concerns the application of different artificial neural network architectures on the classification of multivariate accelerometer time series data into activity classes such as sitting, lying down, running, or walking. There is a strong correlation between increased health risks in children and amount of daily screen time (as reported in questionnaires). The dependency is not clearly understood, as there are no such dependencies reported when the sedentary (idle) time is measured objectively. Consequently, there is an interest from the medical side to be able to perform such objective measurements. To enable large studies the measurement equipment should ideally be low-cost and non-intrusive.

The report investigates how well these movement patterns can be distinguished given a certain measurement setup and a certain network structure, and how well the networks generalise to noisier data. Recurrent neural networks are given extra attention among the different networks, since they are considered well suited for data of sequential nature. Close to state-of-the-art results (95% weighted F1-score) are obtained for the tasks with 4 and 5 classes, which is notable since a considerably smaller number of sensors is used than in the previously published results. Another contribution of this thesis is that a new labeled dataset with 12 activity categories is provided, consisting of around 6 hours of recordings, comparable in number of samples to benchmarking datasets. The data collection was made in collaboration with the Department of Public Health at Karolinska Institutet.

Sammanfattning

Inom ramen för uppsatsen testas hur väl rörelsemönster kan urskiljas ur accelerometerdata med hjälp av den gren av maskininlärning som kallas djupinlärning; där djupa artificiella neurala nätverk av noder funktionsapproximerar mappandes från domänen av sensordata till olika fördefinerade kategorier av aktiviteter så som gång, stående, sittande eller liggande. Det finns ett intresse från den medicinska sidan att kunna mäta fysisk aktivitet objektivt, bland annat eftersom det visats att det finns en korrelation mellan ökade hälsorisker hos barn och deras mängd daglig skärmtid. Denna typ av mätningar ska helst kunna göras med icke-invasiv utrustning till låg kostnad för att kunna göra större studier.

Enklare nätverksarkitekturer samt återimplementeringar av bästa möjliga teknik inom området Mänsklig aktivitetsigenkänning (HAR) testas både på ett benchmarkingdataset och på egeninhämtad data i samarbete med Institutet för Folkhälsovetenskap på Karolinska Institutet och resultat redovisas för olika val av möjliga klassificeringar och olika antal dimensioner per mätpunkt. De uppnådda resultaten (95% F1-score) på ett 4- och 5-klass-problem är jämförbara med de bästa tidigare publicerade resultaten för aktivitetsigenkänning, vilket är anmärkningsvärt då då betydligt färre accelerometrar har använts här än i de åsyftade studierna. Förutom klassificeringsresultaten som redovisas bidrar det här arbetet med ett nytt inhämtat och kategorimärkt dataset; KTH-KI-AA. Det är jämförbart i antal datapunkter med spridda benchmarkingdataset inom HAR-området.

Contents

Chapter 1

Introduction

1.1 Background and motivation

This thesis concerns the application of different artificial neural network (ANN) architectures on the classification of movement patterns from body-worn multivariate sensor time series data into classes such as sitting, sleeping, running, or walking. It is investigated how well these patterns can be distinguished given a certain measurement setup and network structure, and how well the networks perform on self-reported (and thus, "noisier" – in terms of the labels and in the non-laboratory setting of the recordings) data. Recurrent neural networks are given extra attention among the different networks, since they are considered well suited for data of sequential nature [1].

There is a strong correlation between increased health risks in children and the amount of daily screen time (as reported by parents, teachers or the subjects themselves). However, the dependency is not clearly understood, as there are no such dependencies reported when the sedentary (idle) time is measured objectively [2]. Hence, the Department for Public Health at the Karolinska Institute is interested in such an objective measurement method of physical activity. To enable large studies the measurement equipment should ideally be low-cost and non-intrusive, which motivates the use of only two accelerometers in the experiments presented here.

A large survey [2] over studies made on sedentary behaviour and health indicators in youth reports that only 15 out of 232 studies used direct measurements, out of which 14 were performed using accelerometers and one using a monitoring equipment. Non-direct measurements most often consisted of parent-, teacher- or self-reported questionnaires. When accelerometers were used, only the level of intensity of the activity was interpreted from the data and not which specific activity it might correspond to [3]. This is done by somewhat arbitrarily choosing "cut-off" thresholds for different intensity levels in the data. Consequently, a means to objectively measure the proportion of time spent in a sedentary or other state is called for.

The thesis is part of a collaborative project between KTH (Sofia Broomé, supervised by Josephine Sullivan) and the Karolinska Institutet Department of Public Health (Petra Thelin from the Linköping University Medical School, supervised by Daniel Berglind, a post-doc at Karolinska). The aim of the thesis has been to assess whether this kind of activity detection would be possible for a larger study, by designing neural network models for classifying human motion patterns trained with data from accelerometers that subjects have worn while performing everyday activities in a controlled setting. Furthermore, an important part of

the work has been to compare what kinds of architectures are appropriate for datasets of different dimensionality.

1.2 Research question

Is a recurrent deep learning approach suitable for classifying motion patterns (in particular, sedentary vs. non-sedentary patterns) in accelerometer data, and how does it perform with regard to measurement setup (e.g. number of sensors), network architecture, and generalizability? Do measurements from just two accelerometers suffice to classify activities?

1.3 Related work

1.3.1 Human activity recognition

Human activity recognition (HAR) is an application of machine learning that like many others recently is making the transition from hand-crafted feature engineered techniques toward deep end-to-end learning. Zhang *et al.* [4] show that deep learning methods perform better at HAR-tasks than the classical feature-based machine learning methods. The two main references for the work at hand are two 2016 articles; [5] by Hammerla *et al.* and [6] by Ordoñez *et al.*, which both investigate the performance of various deep architectures on HAR problems, where the raw sensor data (at most with some whitening pre-processing applied on it) is used as input.

[5] is currently the state-of-the-art on the Opportunity [7] challenge benchmark 18-class task for gestures, having surpassed [6] with 1% in weighted F1-score, whereas [6] still retains its first place on the Opportunity challenge 5-class task for static or periodic motion.

Hammerla *et al.* [5] state as one of their main purposes that they want to report in an unbiased fashion about the parameter search that preceded their results, something they think is lacking in virtually all previous publications on the topic of deep learning in HAR (perhaps alluding to [6] which article was published a few months prior to [5]). A similar study, transparent about its parameter search regarding general applications of recurrent deep learning, albeit not on HAR problems, is the one by Greff *et al.* from 2015 [8].

In [5], 4 different deep architectures are tried out on three different datasets; a deep feed-forward neural network (DNN), a convolutional neural network (CNN), a long short-term memory network (LSTM) and bidirectional LSTM on the respective HAR datasets PAMAP2 [9], Daphnet Gait (DG) [10] and Opportunity (OPP) [7]. The authors conclude that the DNN is the most sensitive to hyperparameter settings among the models and that one thus is more likely to reduce parameter search time with the other ones. Their best performing architecture on the Opportunity gestures task is the bidirectional LSTM at 92.7% weighted F1-score. Depending on the model, it is reported that different categories of model parameters such as learning, regularisation and architecture have different influence.

In Ordoñez *et al.*'s article [6], different deep architectures are again tested, both on the Opportunity gestures and the Opportunity static/periodic motion task. Notably the best performing one is the authors' own architecture that they've named the DeepConvLSTM. The DeepConvLSTM consists of 4 convolutional layers that are intended to learn and extract an abstract representation of the data and that are followed by two recurrent LSTM layers, taking time dependencies into account. As a baseline, the DeepConvLSTM is compared in

the article to a corresponding six layer network that has two densely connected layers at the end instead of the two LSTM layers of the DeepConvLSTM.

The DeepConvLSTM obtains 91.5% and 86.6%on gestures and 89.5% and 93.0% on the static/periodic motion task (without and with the *null class*, respectively). The null class is the data that embeds the labeled activities during a recording session. Furthermore, on the gestures task the network is tested for parts of the dataset corresponding to different kinds of apparatus measuring different modalities. It is found by the authors that the fusing of multimodal sensors improves performance. The experiment on the gestures task made with only accelerometers (5 sensors, with 15 sensor channels in total) obtains 68.9% weighted F1-score.

A third recent article on the HAR subject is *A Comparison Study of Classifier Algorithms for Cross-Person Physical Activity Recognition* by Saez *et al.* [11]. They collect data from controlled activities using three IMUs, as well as other simultaneous measurements like temperature and heart rate, all in all resulting in a highly multimodal dataset. By pre-processing their data with signal methods and using extra randomized trees as classifier, they obtain 96% average accuracy on their test set. This is state-of-the-art in general for HAR on test sets where the test subjects are entirely left out from the training data. Their classification is done over 12 classes.

1.3.2 Recurrent neural networks and LSTMs

The Long Short Term Memory network was introduced 1995 in a technical report by Hochreiter and Schmidhuber [12], and was later refined in an article by the same authors published in 1997 [13]. The bidirectional LSTM network was introduced by Graves and Schmidhuber in an article about speech recognition in 2005 [14]. The crucial *forget gate* which is now considered part of the standard LSTM structure was introduced in 2000 by Gers *et al.* [15]. A systematic study of RNNs and LSTMs applied on supervised sequence labelling was made by Alex Graves and published as a text book by Springer [16]. Theory regarding RNNs and LSTMs is dealt with in section 2.2.4.

1.4 Outline of the report

In the theory section I will present background theory for how neural networks function. The method section contains details about the two datasets used, as well as about the experiments I ran, and some practicalities surrounding them. The combined results and discussion section presents the results while at the same time commenting on and discussing them. Last comes a short conclusion section about my main results, followed by the bibliography and appendix.

Chapter 2

Theory

2.1 Preliminaries

The following are some useful machine learning concepts that will appear in the report.

Cross-entropy The cross-entropy relates to the Kullback-Leibler divergence, a measure of dissimilarity of two probability distributions P and Q, which is defined as:

$$\mathbb{KL}(P||Q) \triangleq \sum_{k=1}^{K} P_k \log \frac{P_k}{Q_k}, \tag{2.1}$$

for discrete distributions. The k represents a specific possible outcome of a distribution. The Kullback-Leibler divergence may be rewritten as

$$\mathbb{KL}(P||Q) = \sum_{k=1}^{K} P_k \log P_k - \sum_{k=1}^{K} P_k \log Q_k = -\mathbb{H}(P) + \mathbb{H}(P, Q). \tag{2.2}$$

The cross-entropy is the second term in equation (2.2). The first term is equivalent to a system's own entropy ($\mathbb{H}(P) = \mathbb{H}(P, P)$), from which we can see that the KL divergence is 0 for two equal distributions. The cross-entropy can be interpreted as the average number of bits needed to encode data coming from a source with distribution P when we use model Q to define our symbolic language. Accordingly, $\mathbb{H}(P)$ is the expected number of bits required when just using the true model. [17]

Epochs and iterations An epoch is a training session where the network has seen and possibly adjusted to all the training examples. Commonly, neural networks are trained for several epochs. A learning iteration in the context of neural network learning usually means one gradient-based update of the parameters. Depending on how you choose to train your network, this can happen once per epoch or many times per epoch (in that case with smaller parts of the training data). The latter method is called *minibatch* training.

Features Input data coordinates.

Likelihood function In statistics, the likelihood function (often referred to as just the likelihood) is a function of the parameters of a probability distribution given the data.

Log-likelihood The log-likelihood is the natural logarithm of the likelihood function, often applied to be able to work with a sum instead of a product in order to avoid numerical computational issues like underflow.

Maximum likelihood estimation A method of estimating the parameters of a statistical model given the data, where the parameters are chosen so as to maximize the likelihood function.

Overfitting The notion of when a learning system has adjusted so much to the noise in its training data that its ability to generalize even to a test set generated from the same distribution has deteriorated. The extreme case is when the system memorizes the entire training set. Overfitting is related to the very central concepts of variance and bias in machine learning (between which two there is often a trade-off). Increased variance and low bias is typical for an overfitted model, whereas increased bias and low variance is typical for an underfitted model. It is common to want to find an optimal point between these two extremes.

Regularization Regularization of a machine learning algorithm means to prevent it from overfitting, which can be done in many different ways. A common method of regularization is to restrict how large the parameters are allowed to grow (in accordance with the principle of simplicity of Occam's razor), by adding the norm of the parameters to the cost function. In deep learning, dropout is a common regularizer, which means that one randomly sets the output of certain nodes to 0 for every epoch.

Supervised learning The branch of machine learning where the system learns by feeding the computer a key of the right answers while training. The program then continuously updates its parameters in order to conform to this key. This is in contrast to *unsupervised* learning where no such key exists and the program has to orientate "blindly" in the data and discover patterns on "its own", and to *reinforcement* learning where the system is fed a scalar reward for every training iteration telling it how well it did, in relation to a particular goal.

2.2 Deep learning

During the past half decade, there has been a veritable surge of research in the subject of so called deep learning. Commonly, it is said that the wave gained decisive momentum in the field of computer vision in 2012 when Krizhevsky *et al.* [18] considerably beat the best results at the time with their network AlexNet on the ImageNet ILSVRC-2010 contest. There, the task is to classify 1.2 millions of high-resolution images into 1000 classes [19]. Handcrafted feature extraction methods within image recognition such as SIFT [20] or HOG [21] suddenly seemed outdated, compared to the deep CNN employed by the Krizshevsky team.

The idea of deep learning is to let the computer learn from experience (data) and to make sense of the world in terms of concepts that build on each other hierarchically (much like humans learn starting from shape, color or sound and going toward richer experiences). This reduces the need for the programmer to manually specify features corresponding to for example, in the context of our study, different interpretation levels of the complex skeletal expressions that different forms of locomotion take. The hierarchy of concepts that a deep learning architecture consists of is reflected in the layers of an ANN. When a network has

many layers we call it deep. [1]

2.2.1 ANNs and the multi-layer perceptron

An artificial neural network is a differentiable function approximator in the capacity of a computational graph. The computational graph is typically constituted of affine transformations followed by nonlinear functions, stacked on each other in a chain. What function the graph approximates is decided by its architecture (meaning the number of layers and the number of computational units in the layers) and by its parameters. Usually, the architecture is decided by an expert (although a recent blog post [22] is about the company in question's recent effort of having a child network learn new and better architectures automatically) and the parameters are what is computed during learning [23]. The learning phase involves input data, and some goal of what the network should output, according to which the parameters are adjusted.

Quick history

ANNs in their current form (which crucially means *with* nonlinearities and *with* the associated back-propagation algorithm for training, although early forms of both had appeared starting in the 1960s) were introduced during the 1980s [24]. Linear kinds of perceptrons and artificial neurons had been around starting from the 1940s. The name ANN comes from that these networks initially were intended as models of biological learning, notably under the *connectionist* paradigm that arose in the 1980s within the field of cognitive science. [1]

Connectionists recognized the potential of achieving intelligent behaviour via many small computational units working together in a network, and freely highlighted the similarities between brains and computers (though inspiring, this handwaving tradition carries on into today's deep learning in computer science).

During the 1990s, companies using AI technologies like neural networks started to make promises to investors that they couldn't quite fullfil, due to lack of computational power and lack of training data [1]. Partly because of this, ANNs successively lost their popularity, both in industry and in academia. Their reentry on the scene came in the early 2010s when the computational power had increased, in part thanks to the broad introduction of GPUs as parallel computing tools and partly as the amount of available training data had increased. They are now state-of-the-art for tasks in various applications such as vision, speech and translation. [1], [25]

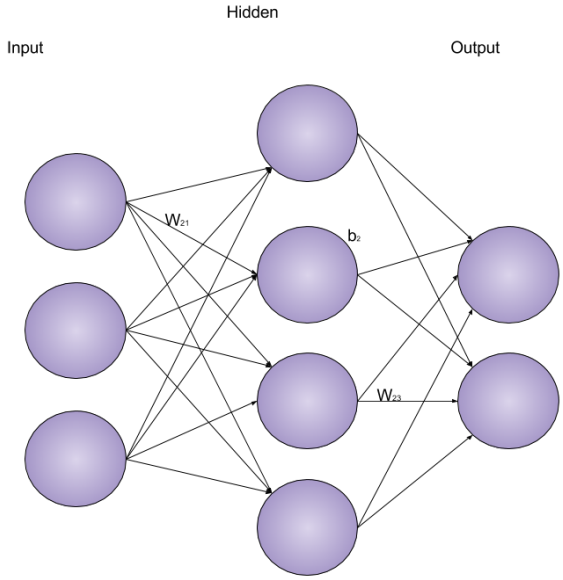
MLP basics

The core of deep learning can be explained by a multi-layer perceptron (MLP), also called a deep feedforward network [1], which is a simple artificial neural network where all nodes in one layer are connected to all the nodes in the next layer, and there thus is a full weight matrix and bias vector between each of the layers. The weight matrix is of the dimension $h_k \times h_{k-1}$, if h_k is the number of units in layer k, and the bias vector of the same layer is of the dimension $h_k \times 1$.

This means that each edge incoming to a layer k has an independent parameter, W^k_{ji}, belonging to the directed edge going from node i in layer $k-1$ to node j in layer k, and each node in layer k has a bias parameter b^k_i. The nodes and layers that are in between the input and output layer are called hidden. See figure 2.1. Incoming edges are summed up at

the arrival node, where the corresponding bias vector element is also added to that node's value, whereupon some nonlinear function typically is applied on the entire scalar value of the sum.

Figure 2.1: *An example of a simple MLP*



Mathematically, an MLP is thus just a function mapping some set of input values to some corresponding set of output values, through layers of functions stacked on top of each other. This is the case for the more sophisticated networks that are described later in this chapter as well although their richer structure sometimes on an intuitive level obscures the fact that they really are "mere" function compositions.

The algorithm for forward propagation of information through an artificial neural network for the case of per-example training is summarized in algorithm 0.

Algorithm 0: Forward propagation [1]

Require: Network depth, l (total number of layers, including input and output layers).
Require: $\mathbf{W}^{(k)}, k \in \{1, \ldots, l\}$, the weight matrices of the model
Require: $\mathbf{b}^{(k)}, k \in \{1, \ldots, l\}$, the bias parameters of the model
Require: $\mathbf{x}$, the input to process
Require: $\mathbf{y}$, the target output

$\mathbf{h}^{(0)} = \mathbf{x}$
for $k = 1, \ldots, l$ **do**
$\mathbf{a}^{(k)} = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}$

if $k < l$ **then**
$\mathbf{h}^{(k)} = f(\mathbf{a}^{(k)})$
else
$\mathbf{h}^{(k)} = \text{softmax}(\mathbf{a}^{(k)})$
end if

end for

$\hat{\mathbf{y}} = \mathbf{h}^{(l)}$
$J = L(\hat{\mathbf{y}}, \mathbf{y}) + \lambda\Omega(\theta),$

---

where $L(\hat{\mathbf{y}}, \mathbf{y})$ is what's called a *loss function* and $\Omega(\theta)$ is a parameter penalty regularizer, controlled by some scalar λ, and f is some nonlinear function. $\mathbf{h}^{(k)}$ is a vector whose elements denote the levels of response (or so called "activation") for the nodes of layer k, and more specifically $\mathbf{h}_i^{(k)}$ denotes the activation for unit i in layer k.

At layer l, the output layer, the nonlinear function is often the *softmax function*, like in algorithm 0. The output from the softmax function (the activation $\mathbf{h}^{(l)}$) is a vector with elements between 0 and 1 summing to one. In classification tasks, this is interpreted as a probability distribution over a discrete variable with the classification labels as possible values. The output for one element from the softmax function is as follows

$$\text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}. \tag{2.3}$$

W_{ij}^l denotes, as has already been stated, the weights associated with the edge between unit j in layer l and unit i in layer $l+1$. The reason that this subscript notation seems reversed is just to avoid a transpose in stating the activation equation. Figure 2.2 further illustrates the different weights and activations.

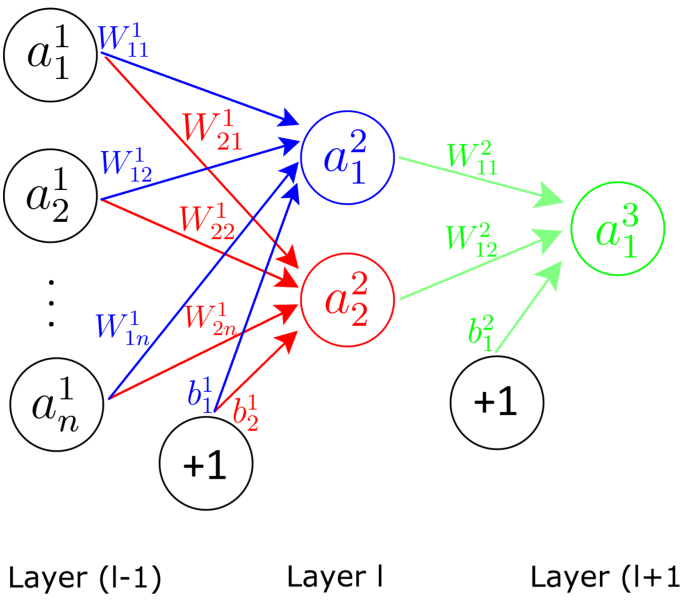


Figure 2.2: *An MLP with three densely connected layers. Figure from [6].*

An example of a hierarchical feedforward neural network used for object recognition in images is in figure 2.3. The features extracted at the hidden layers, visualized in the figure, are

the parts or aspects of the image that the *activation functions* reacted the most strongly to for each layer. As can be seen, the weights of the different layers have adjusted to filter for particular patterns. Activation functions are the nonlinear functions that are used to compute the ultimate values of the hidden nodes of a network.

A regular MLP is limited by the fact that it assumes that the weights in the network are independent. As we've seen in figures 2.2 and 2.1, between two layers in a fully-connected (dense) MLP there are separate weights between every pair of nodes. These are optimized separately. A computer vision example can illustrate a consequence of this.

If an MLP network at some layer detects a face in the top right corner of the input image, the weights of the network corresponding to that activation will be specific to the location of the face, since they are able to specifically adjust to structure (or noise!) in the image at hand. The difference with a CNN in that regard is that the same (small) set of parameters of a convolutional layer are slid across the entire image and can thus react to a face at any location of the image, if it once has been trained to recognize a face. This is elaborated on in section 2.2.3.

And so, the problem is not that an MLP cannot find spatial or temporal structure in input data. It's rather that what it discovers is *too* spatially or temporally dependent, since a pattern needs to be seen at a particular location in order to be recognized.

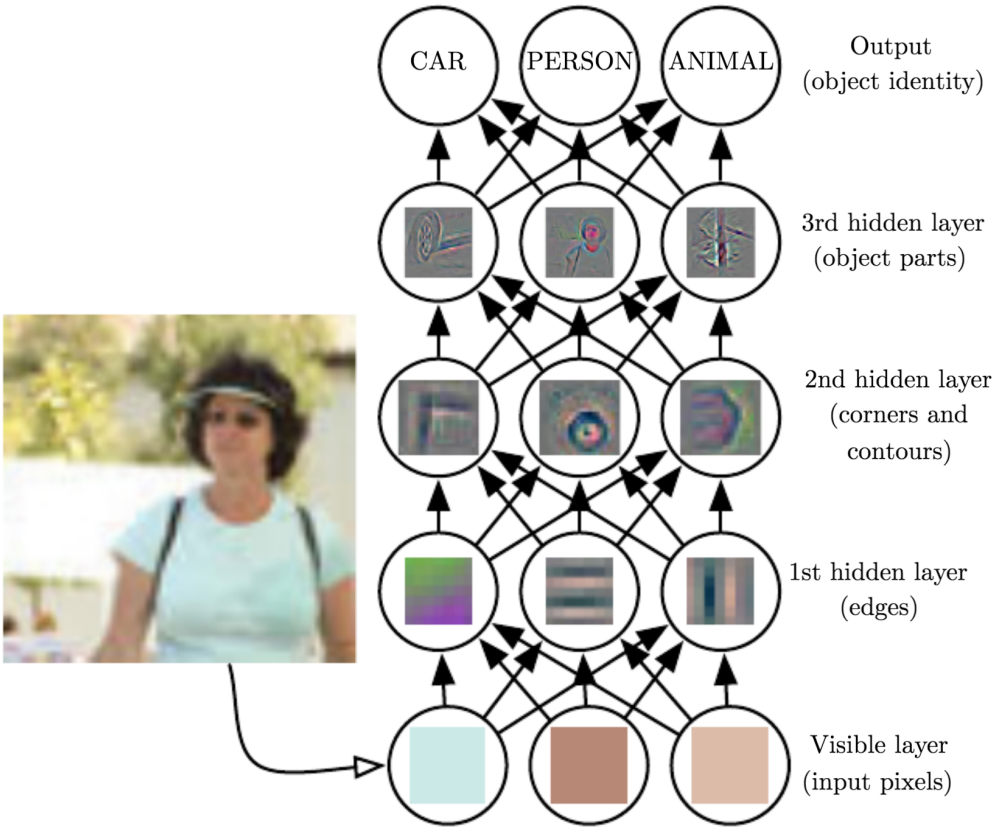


Figure 2.3: *A scheme of a typical computer vision feedforward network for object recognition and classification. Image from [1].*

In keeping with the notation in [1], we refer to the function we wish to approximate as f^*.

That is, the function that for instance could map a part of a multivariate time series taken from an accelerometer that someone wore on their waist for some time to the category "walking". It could also be the function that could map an English sentence to its Swedish translation, or an image of a flower to the name of its exact species. Our understanding of the world can often be interpreted as functions, and many of these functions can be learned by artificial neural networks.

In the case of classification, the as it were "real world" mapping of an entity x to a category y can then be written as $y = f^*(x)$. Our MLP would define the mapping $\mathbf{y} = f(\mathbf{x}; \theta)$ and have the goal to try to learn the set of parameters θ in order to come as close as possible to the true function f^*.

It's been proven [26] that a network structure with one hidden layer, nonlinear activation and a sufficient number of these hidden units can approximate any smooth function to an arbitrary degree. These networks are thus *universal approximators*, in their mapping of input vectors to output vectors, which is what makes them so useful for tasks in artificial intelligence. On the other hand, the sufficient number of hidden units of [26] is not guaranteed to be manageable computationally [27]. As is pointed out by Lin *et al.* in [27], the reason that a lot of neural network function approximations however indeed seem to work for a variety of tasks [25] is that the class of functions we are actually interested in is tiny, and essentially of low dimensionality, compared to the total collection of estimable functions. A photo of a galaxy consisting of a million pixels can really be understood as having been generated by a probability distribution determined by the standard model of particle physics which has only 32 parameters. This is furthermore in analogy with the "manifold assumption", a concept often encountered in machine learning literature, where one assumes that the data in question really lies on or close to a low(er) dimensional manifold than the space it is embedded in [28].

2.2.2 Training an artificial neural network

Gradient descent and the objective function

The learning of the network is commonly done by the process of *gradient descent*. Associated with our two functions f^* and f is the *objective function*, that we want to optimize for our purposes (either by minimizing or maximizing it, depending on how we cast the problem).

Gradient descent, in the case of minimization, proceeds by taking small steps along the opposite direction of the objective function's gradient since it points upwards in the direction of the steepest incline. The size of the steps taken along the gradient is called the learning rate, α, a tuning parameter highly influential to the optimization process, according to both [6] and [5]. The updated point θ' suggested by the most general gradient descent algorithm is

$$\theta' = \theta - \alpha \nabla_\theta f(\theta). \tag{2.4}$$

The learning rate α doesn't have to be fixed throughout the learning process. Typically, to be efficient, you want to take large steps along the cost function surface in the beginning and smaller when you approach a local optimum, in order to not accidentally "walk past" the optimum. A common thing to add to the gradient descent update is what's called *momentum*. Momentum is designed to increase the speed of the learning by incorporating the previous gradients into the updates. This makes the trajectory more robust to noisy gradients or to

obstacles like curvature along the way. The update equations are the following (after having initialized $\mathbf{v}$ and θ):

$$\mathbf{v}' = \alpha\mathbf{v} - \epsilon\nabla_\theta f(\theta) \tag{2.5}$$

and

$$\theta' = \theta + \mathbf{v}. \tag{2.6}$$

Other common examples of optimization algorithms with adaptive learning rates are RMSProp, Adagrad and Adam. Adagrad and RMSProp are similar. Where Adagrad divides the current gradient with the accumulated squared gradient in the update step of every iteration, RMSProp has a parameter to adjust how much weight to put on the current gradient and on the accumulated squared gradient division in the update. Adam uses elements similar to both RMSProp and momentum. Its name comes from "adaptive moments" [1]. The algorithm for Adam is given in appendix A.4.1.

I will here refer to the total objective function as $\mathcal{E}(\theta)$ and the per-example objective function as $L(\mathbf{x}, y, \theta)$. L stands for loss – the objective function is sometimes called the cost function or loss function.

The objective function used in deep learning is often the average of the per-example loss function for the set of training examples [1]. For example, the negative conditional log-likelihood of the total training data can be written as

$$\mathcal{E}(\theta) = \mathbb{E}_{\mathbf{x},y\sim\tilde{P}_{data}} L(\mathbf{x}, y, \theta) = \frac{1}{m}\sum_{i=1}^{m} L(\mathbf{x}^{(i)}, y^{(i)}, \theta), \tag{2.7}$$

given that $L(\mathbf{x}, y, \theta) = -\log\big(P(y|\mathbf{x};\theta)\big)$. $\tilde{P}_{data}$ here symbolizes the true generating data distribution, and P the estimated generating data distribution.

The conditional log-likelihood is at the foundation of supervised learning, since supervised learning really is about estimating the quantity $\tilde{P}(\mathbf{y}|\mathbf{x}, \theta)$ in order to predict $\mathbf{y}$ given $\mathbf{x}$ [1]. If we let $\mathbf{X}$ represent all of our input data and $\mathbf{Y}$ the targets associated with that data, then the estimator of the conditional maximum likelihood is

$$\theta_{ML} = \arg\max_\theta P(\mathbf{Y}|\mathbf{X};\theta). \tag{2.8}$$

Making the assumption that all data examples are identically and independently distributed (i.i.d.), this can instead be written as

$$\theta_{ML} = \arg\max_\theta \sum_{i=1}^{m} \log(P(\mathbf{y}^{(i)}|\mathbf{x}^{(i)};\theta)), \tag{2.9}$$

where the step of factorizing the independent probabilities in equation (2.8) and subsequently taking the logarithm of this product was omitted. A maximum likelihood estimation is equivalent to the cross-entropy between the training data distribution and the model distribution, also called the negative log-likelihood.

The exact form of $\log(P(\mathbf{y}^{(i)}|\mathbf{x}^{(i)};\theta))$ changes according to the assumptions underlying different models. In the case where we assume a Gaussian distribution for $P(\mathbf{y}|\mathbf{x};\theta)$ (i.e. that $P(\mathbf{y}|\mathbf{x};\theta) \sim \mathcal{N}(\mathbf{y}; f(\mathbf{x};\theta, \mathbf{I}))$, we obtain the mean squared error cost as our objective;

$$\mathcal{E}(\theta) = \mathbb{E}_{\mathbf{x},y\sim\tilde{P}_{data}} ||(\mathbf{y} - f(\mathbf{x};\theta)||^2 + \text{const.} \tag{2.10}$$

A recurring problem in neural network design is to keep the gradient of the objective function large enough to guide us in how to orientate during optimization. If it becomes too large or too small, it can't do its job. The vanishing gradient problem is due to that the output units' activation functions might saturate. This can be understood by picturing a common activation function - the sigmoid function, and what its derivative will look like for large or small values (0). See figure 2.4.

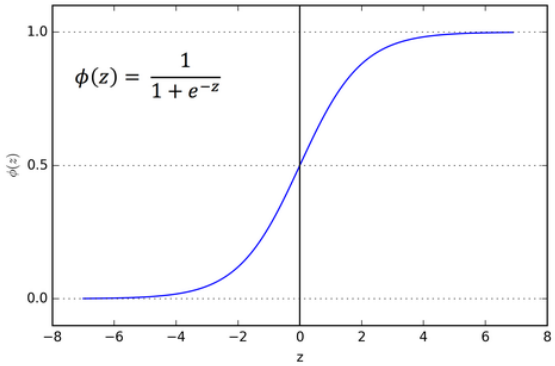


Figure 2.4: *A typical sigmoid function.*

The handy thing about the negative log-likelihood is that the exponential function in the sigmoid activation function (or any other activation function containing an exponential function, which is almost always the case for the output neurons for classification tasks since we want our output as a probability distribution between 0 and 1 over the possible classes) is cancelled by the log in equation (2.9).

Minibatch training

Since computing

$$\nabla_\theta \mathcal{E}(\theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta L(\mathbf{x}^{(i)}, y^{(i)}, \theta) \tag{2.11}$$

has the computational cost of $\mathcal{O}(m)$, it is most common to compute an estimate of the gradient as the average gradient over a *minibatch*, or just batch, of training examples, $\mathbb{B} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(m')}\}$, at every learning iteration, until the whole set of training samples has been worked through. The members of the different batches are either drawn with uniform probability from the training set at every new training epoch or the data is just chopped up in chunks of the desired batch size, with their internal order kept intact.

Back-propagation

For feedforward neural networks, the actual gradients of the training examples are computed using the backpropagation algorithm. Consider the forward information flow through a feedforward network: we can refer to this as forward propagation. As we have seen, the forward propagation produces a scalar cost $\mathcal{E}(\theta)$ for every training iteration. The back-propagation algorithm (sometimes just referred to as backprop) handles the flow of information *back* through the network, starting from the cost, to compute the gradient of the cost function with reference to the parameters. Having already stated that the neural network is a collection of composite functions, the analytical way to solve for $\nabla_\theta \mathcal{E}(\theta)$ would be using the

chain rule. Backprop does this, but more efficiently, by reusing already computed quantities, in a dynamic programming fashion.

Batch normalization

When making an update to a network parameter, we take the gradient of the cost function with reference to that parameter, assuming that all other parameters are kept constant. This is an assumption that works, but is artificial since in most implementations, all parameters are updated simultaneously [1]. Batch normalization is a technique that standardizes the mean and variance of each unit in order to stabilize learning and mitigate the perturbations stemming from the flaws in the gradient estimations. If $\mathbf{H}$ is a matrix where the rows correspond to every training example of a batch, and the column elements each correspond to different activations in a network, the reparametrization is done by

$$\mathbf{H}' = \text{diag}(\boldsymbol{\sigma})^{-1}(\mathbf{H} - \boldsymbol{\mu}), \tag{2.12}$$

where the $\boldsymbol{\mu}$ is a vector containing the mean of each unit and $\boldsymbol{\sigma}$ is a vector containing the standard deviation of each unit.

2.2.3 Convolutional neural networks

What distinguishes a CNN from a regular feedforward network is that it at some point in the network structure contains the linear mathematical operation *convolution*. Usually, a CNN has one or more layers that are referred to as convolutional, which means that instead of regular matrix multiplication they use convolution in the computation of that layer's activation.

A convolution operates on two functions as the integral of their product with one of them reversed and shifted. This results in a third function, exemplified by the following integral:

$$(x * w)(t) = \int_{-\infty}^{\infty} x(a)w(t-a)da, \tag{2.13}$$

or in discrete form:

$$(x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a). \tag{2.14}$$

The function x can be thought of as the input function, and the function w as the kernel or weighting function. In a machine learning setting, we can then refer to the output as a feature map. [1]

Sparse connectivity

A convolutional layer in a neural network differs from a regular fully-connected layer especially in one important sense. The fully-connected layer has, as the name suggests, connections between every input and output node, which means that all of these connections are treated as independent. In contrast to this, a convolutional layer has *sparse* connectivity to its previous layer, something that takes into account the spatial (or temporal) structure of the input data, and can thus have localized groups of the input data relating to particular hidden neurons. This is illustrated in the figure 2.5 below, and is furthermore what allows for the specific object activations of certain neurons in figure 2.3.

Apart from detecting spatial or temporal patterns in the data, sparse connectivity also decreases the number of parameters of a model. Often, the very same kernel is slid across the input, meaning that the same small set of parameters is used for all connections to the next layer.

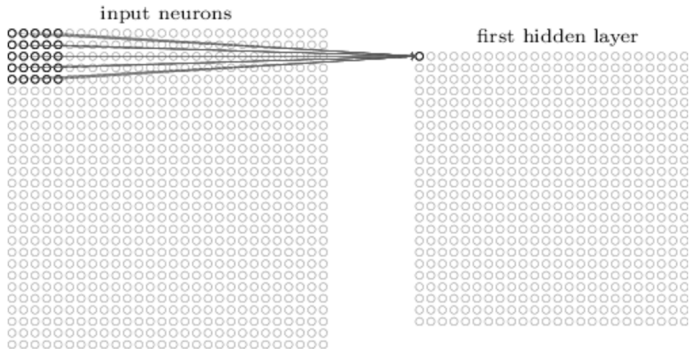
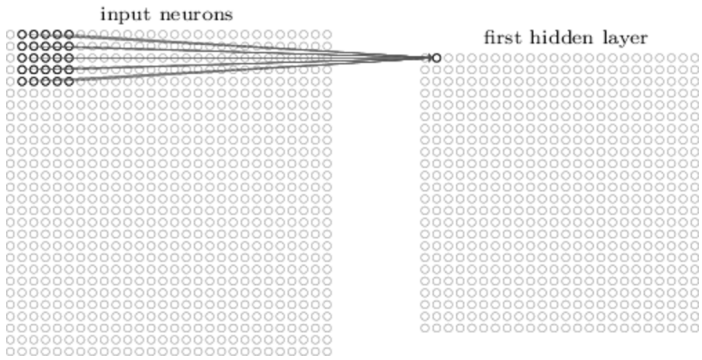


Figure 2.5: *Typical sparse connectivity in a convolutional network layer. Figure from [29].*

In figure 2.5, the kernel has size 5×5, and is slid over the columns and rows of the input data. The kernel can move across the input neurons with a certain step size, or *stride*. In the image below (figure 2.6), the kernel has moved to the right with stride 1, and sends this information to the next hidden neuron in the upper row of the hidden layer.

Figure 2.6: *The convolutional kernel has moved with stride 1, to the right. Figure from [29].*



The figures (2.5 - 2.6) show an example of 2D convolution. This refers to the fact that the kernel is two-dimensional. Depending on the kind of input we send to the neural network, we will use convolutions of different numbers of dimensions. If our data is a time series, we might use 1D convolution, whereas if our input is a color image (meaning a 3D tensor input), we might use a 3D convolution. Make the color image a video and we can add a fourth dimension to our kernel. The following equation is an example of a discrete convolution with a two-dimensional kernel K:

$$(I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n). \tag{2.15}$$

Note the minus sign in the indices of the kernel. This allows the convolution operation to be commutative, which is a useful property in mathematics but one that is often not necessary in

machine learning. For this reason, many implementations of convolutional layers in machine learning libraries instead use the related operation called *cross-correlation*: [1]

$$(I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n), \tag{2.16}$$

which is no longer commutative. In a practical study like this one, equation (2.16) is what will be referred to by the term convolution, following the somewhat loose convention of the online deep learning language.

A convolutional layer is most commonly followed by a non-linearity, for example the rectified linear unit (ReLU) which has had good results for convolutional networks, and by a pooling layer. The ReLU activation is defined by

$$\text{ReLU}(x) = \max(0, x). \tag{2.17}$$

In contrast to the sigmoid activation, the ReLU's derivative doesn't saturate for large values of x.

A pooling layer can do localized statistical operations like taking the max of a number of units, or their average. This can again help reduce the number of parameters of the network, as well as down-sample the input data into a more essential form.

2.2.4 Recurrent neural networks

When there are recurrent connections present between nodes in a network, we call it a recurrent neural network. The idea with cyclical connectivity is to let the network be able to retain information across different time steps and to make connections between the different temporal parts of the data.

Specifically, this is done by feeding the activation of a unit at the previous time step to the same node at the present time step. This type of "memory" makes the RNNs suitable for sequential data, like natural language, time series, or video: anywhere where the separate data points cannot be assumed to be i.i.d.

Just like with the convolutional neural networks, another important aspect of an RNN is its parameter sharing. For a CNN, the parameter sharing was across one time step at a time. For every time step, a kernel of a certain configuration was swept across the input, meaning that temporally, the parameter sharing was *shallow*. In the recurrent case, the parameter sharing is deep in the temporal sense. RNNs can contain a mapping that runs through the entire history of the sequence to be classified, with the same parameters kept throughout that chain. This allows for processing of sequences of varying length. [1], [16]

The recursive equation for one hidden node can be stated like

$$\mathbf{h}_t^l = f(W_{xh}^l \mathbf{a}_t^l + \mathbf{h}_{t-1}^l W_{hh}^l + \mathbf{b}_h^l), \tag{2.18}$$

and the activation of the hidden unit like

$$\mathbf{a}_t^{l+1} = h_t^l W_{ha}^l + \mathbf{b}_a^l. \tag{2.19}$$

This is further illustrated in figure (2.7). The function f is often the $tanh$ function. The superscript l denotes the RNN hidden layer on a stacked architectural layer level, as opposed to the temporal layers built into one hidden RNN layer.

As can be seen in figure (2.7) and in equations (2.18) and (2.19), we are now dealing with three different weight matrices. W_{xh} denotes the input-hidden weight matrix, W_{hh} the hidden-hidden weight matrix, and W_{ha} the hidden-activation weight matrix. The weight matrix discussed for regular feedforward networks is equivalent to W_{xh} here.

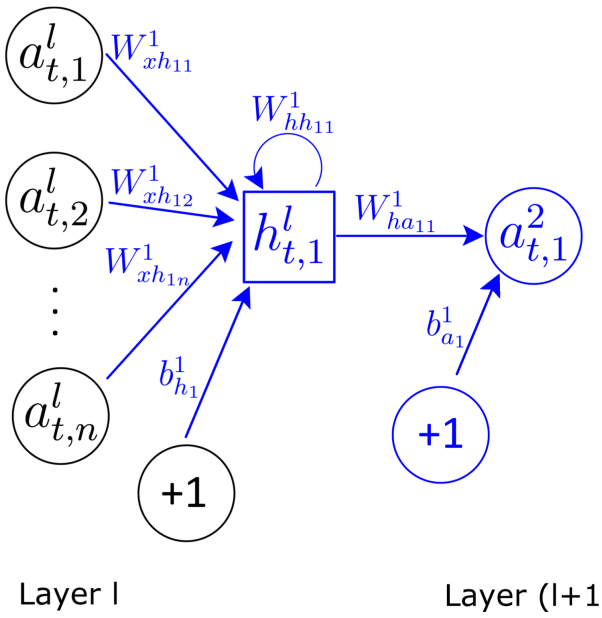


Figure 2.7: *A recurrent neural network (RNN) with two dense layers. Figure from [6].*

The output from a recurrent neural network can be either a sequence itself (implying that there's an output at every time step of the sequence) or just a single output (usually at the last time step of the sequence). For example, in machine translation the output can be a sequence with outputs starting from the first timestep, whereas for example in fixed length sequence classification like in our case, the output is one label at the end of the sequence processing.

The vanishing and exploding gradient problem

The vanishing and exploding gradient problems when dealing with pattern dependencies of longer duration were first formulated by Bengio *et al.* in 1994 [30], and later expanded on in 2012 [31]. The two problems represent two sides of a coin: the computation of the gradient depends exponentially on the number of time steps that are part of the backpropagation, which either makes it [the gradient] potentially vanish after a certain number of backpropagated time steps, or conversely, explode.

Whether it explodes or vanishes depends on the spectral radius of the recurrent weight matrix $\mathbf{W}_{hh}$ – a scalar threshold for explosion can be formulated for the largest eigenvalue of $\mathbf{W}_{hh}$ in relation to the maximum value of the activation function [23]. The recurrent weight matrix appears to the power of $t - k$ in the computing of a product of Jacobians necsssary to obtain the gradient of the loss function for an RNN.

Consider a classical RNN, defined by the equations 2.18 and 2.19. Let θ denote the model parameters and let $\frac{\partial^+ \mathbf{h}_k}{\partial \theta}$ denote what [30] calls the "immediate" partial derivative of the

hidden state $\mathbf{h}_k$ with respect to θ, meaning that we assume no further dependence between $\mathbf{h}_{j<k}$ and θ. If we restrict the model parameters to the outer recurrence defined by the weight matrix W_{hh}, then

$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1<k\leq t} \Big(\frac{\partial \mathcal{E}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial^+ \mathbf{h}_k}{\partial \theta} \Big) \tag{2.20}$$

is the objective function gradient at a certain time step (in back-propagation through time (BPTT) gradients are computed for all time steps). The Jacobian factor $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k}$ is computed as follows:

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \prod_{k<i\leq t} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \prod_{t\geq i>k} \mathbf{W}_{hh}^T \operatorname{diag}\big(\sigma'(\mathbf{h}_{i-1})\big), \tag{2.21}$$

at which point we see the infamous product appear.

Long short term memory (LSTM) networks

When speaking of recurrent neural networks, we can distinguish between short and long term memory. The short term memory can be represented by the activations from the previous timestep, whereas the long term memory can be represented by gradually changing weights throughout the network. [13]

An LSTM network is a recurrent neural network that contains a specific type of hidden node called an LSTM cell. The LSTM cell has an additional *inner* recurrent loop that is self-referencing its own internal state and through that decides what to forget, what to take in and what to output (these last three decisions being in analogy to the computer operations resetting, reading and writing). If a typical (temporally unfolded) chain of recurrent nodes looks like in figure (2.8),

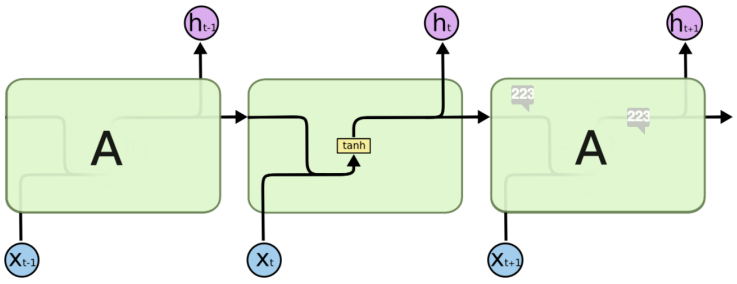


Figure 2.8: *Figure from [32].*

we can contrast the above structure with figure (2.9) and first observe that the LSTM cell has a more complex inner structure. The pink nodes represent pointwise operations (addition or multiplication) and the yellow nodes represent different types of activations (tanh or sigmoid).

In figure (2.9), the *cell state* is the value at the top horizontal edge. To borrow a metaphor from [32], the cell state can be thought of as a conveyor belt, where updates from the forget and input gates are successively merged into it, along the flow of the memory cell. The self-referencing loop has a constant weight of one, whose gradient never vanishes, hence the name "the constant error carousel".

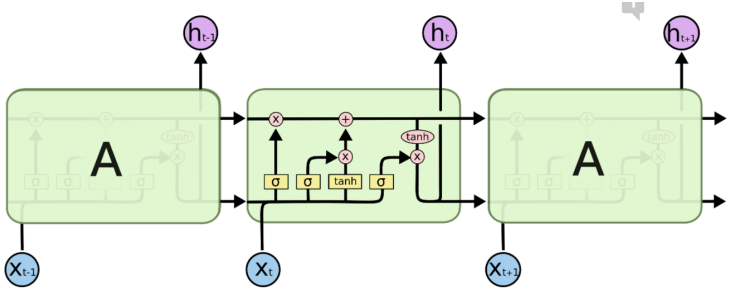


Figure 2.9: *Figure from [32].*

Apart from the input from the (same) hidden node coming from the previous time step, and the usual input from the incoming layer we have three internal *gates* consisting of a sigmoid operation and a pointwise multiplication that each regulate what to forget, what to keep and what to input.

Why is this useful? In classical RNNs, finding long term dependencies in the data turned out to be problematic due to vanishing or exploding gradients during the computing of the back-propagation through time (BPTT) algorithm. Because of that, the long-term memory of the network as discussed above was difficult to keep up.

The LSTM cell is designed to mitigate this. The idea is to keep a constant error flow within the LSTM cells, referred to as the *constant error carrousel* (CEC), keeping the gradients in check.

The way Hochreiter and Schmidhuber describes it in [13], the gradient problem arises from *perturbations*, caused by irrelevant input and output. The input gate is then introduced to hinder irrelevant input to enter the node, and the output gate similarly aims to prevent irrelevant output to exit from the node.

In the 1997 article [13], the LSTM was presented with just the input and output gates; the forget gate wouldn't be introduced until some years later by other authors.

The LSTM gate equations are stated below. The forget gate is denoted by f, the input gate by g, the *cell state* by s and the output gate by q. These each have their own recurrent weight matrices, denoted by W^f, W^g, W^q, as well as biases denoted by b^f, b^g, b^q, and weight matrices for the regular inputs denoted by U^f, U^g, U^q. As usual, h denotes the output from the entire hidden node, here exemplified by one complete LSTM cell. The W, b, U without superscripts denote the outer recurrent weights, biases and input weights into the LSTM cell.

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f \, x_j^{(t)} + \sum_j W_{i,j}^f \, h_j^{(t-1)}\right) \tag{2.22}$$

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\left(b_i + \sum_j U_{i,j} \, x_j^{(t)} + \sum_j W_{i,j} \, h_j^{(t-1)}\right) \tag{2.23}$$

$$g_i^{(t)} = \sigma\left(b_i^g + \sum_j U_{i,j}^g \, x_j^{(t)} + \sum_j W_{i,j}^g \, h_j^{(t-1)}\right) \tag{2.24}$$

$$h_i^{(t)} = \tanh\left(s_i^{(t)}\right) q_i^{(t)} \tag{2.25}$$

$$q_i^{(t)} = \sigma\left(b_i^q + \sum_j U_{i,j}^q \, x_j^{(t)} + \sum_j W_{i,j}^q \, h_j^{(t-1)}\right) \tag{2.26}$$

Since σ denotes the sigmoid function, the values of f, g and q are all between 0 and 1.

Chapter 3

Method

To perform human activity recognition on accelerometer data, the classifier models used were recurrent and non-recurrent neural networks with a varying number of layers and parameters. An introductory classification with Hidden Markov Models (HMMs) was also done to assess whether neural networks were going to be necessary for low-dimensional data.

The neural networks were implemented using both the Python library Tensorflow [33] and its high-level relative Keras [34] (which can be run with a Tensorflow or Theano [35] backend, I used TensorFlow) and were tested on the report's own collected dataset that we refer to as the KTH-KI Accelerometer Activity dataset (KTH-KI-AA) as well as on the open dataset Opportunity. The HMM was implemented using scikit-learn [36], another Python library. The datasets differ in dimensionality; KTH-KI-AA has 6 features where Opportunity has 113, and they will both be described in more detail below. The performance of baseline models and models with richer structure was explored on the two datasets.

At the end of February, the KTH-KI-AA data was collected and labelled for the project at the Karolinska Institute by me and the medical student Petra Thelin. We did the data recordings together for the adult population, then Petra did the recordings of the child population the day after and I then put together the dataset with sets of labels of different granularity.

The recordings were made during two days where subjects wore two triaxial accelerometers, one on the waist and one on the dominant wrist, while performing controlled everyday activities like reading a book or washing the dishes. A noisier type of data collection was also made in collaboration with Karolinska: 4 subjects wore the same accelerometer setup during two full days in their everyday lives. The subjects reported their activities and corresponding intensity levels in paper questionnaires, on a bi-hourly basis.

The experimental setup throughout the work with this thesis was mainly driven by questions regarding what types of architectures, activity category labelling and test subject populations are suitable to obtain generalising and accurate measurements of human activities recorded in few sensor dimensions.

3.1 Body-worn sensor datasets for human activity

3.1.1 The Opportunity dataset

In 2011, a challenge was set out by the 7th Framework Program of the European Commission under the Information and Communication Technologies theme [37], to benchmark recogni-

tion of human activity. A dataset was provided openly on the web of around 6 hours of recordings in total of 4 subjects performing various tasks in a daily living scenario, with sensors of different modalities integrated in the environment (in surrounding objects and on the body). More specifically, the dataset for the benchmark challenge contains 676 713 data points (6.27 hours) with the null category included, and 559 928 data points without the null category (5.18 hours).

The measurements in the dataset were made of both activities of daily living (ADL) and so called drill sessions. The ADL recordings were made of the subjects performing listed activities without restrictions (for example checking ingredients and utensils in the kitchen, preparing and drinking a coffee or eating a sandwich, cleaning up). During the drill sessions on the other hand, the subjects performed 20 repetitions of a predefined sorted set of 17 activities. There are two available sets of labeled classes for the recordings: both static or periodic activities such as standing, walking or lying down as well as specific sporadic gestures such as the one connected to drinking from a cup. [6]

The Opportunity Challenge only included the body-worn sensors in the data, and the experiments of this thesis on Opportunity are also performed with exclusively that part of the data, comprising in total 113 individual sensor channels.

Measurement equipment

The rich wearable sensor family employed in the Opportunity data is composed of the following apparatus: 5 commercial RS485-networked XSense inertial measurement units (IMU) included in a custom-made motion jacket, 2 commercial InertiaCube3 inertial sensors located on each foot and 12 Bluetooth acceleration sensors on the limbs. The IMUs are each multimodal, consisting of a 3D accelerometer, a 3D gyroscope and a 3D magnetic sensor. The sample frequency for all of the feature channels is 30 Hz. [6]

Training, validation and testing

The division of the dataset into a training set and a test set was done according to the rules of the Opportunity benchmarking challenge. The rules said that the test set should consist of the ADL4 and ADL5 recordings from subjects 2 and 3. In the challenge, one was thus free to choose whatever portion of the training data as validation set. Here, I followed Ordoñez *et al.*'s example, using subject 2 and 3's ADL3 recordings.

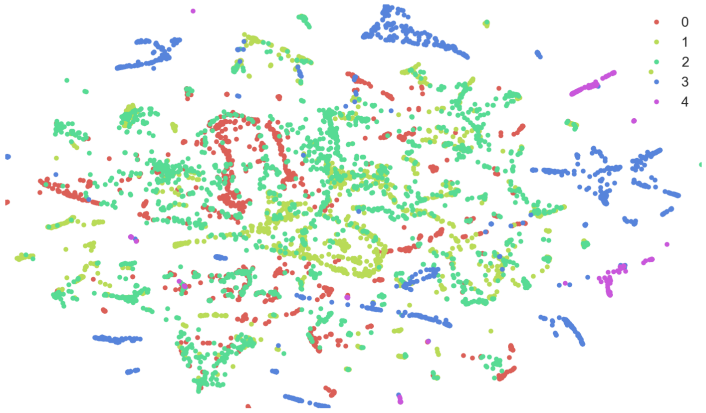
In order to give an idea of to what extent the different classes of the dataset's static motion labelling are separable, I show 2-dimensional t-SNE-projections of the test set with and without the null category in figure 3.1.

Pre-processing

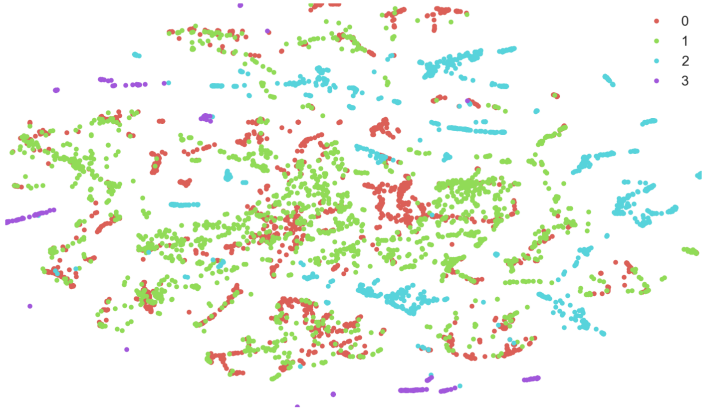
The Opportunity dataset is preprocessed so that all feature channels are normalized to the interval $[0, 1]$. The reason for this pre-processing is that I followed [6]'s example. It is not stated in [5] whether they pre-processed the Opportunity data for their experiments.

3.1.2 The KTH-KI Accelerometer Activity dataset

Together with Petra Thelin, data was collected at Karolinska Institutet in February 2017 during two days. Subjects in a laboratory setting wore two triaxial accelerometers, one on the hip and one on the dominant wrist. An accelerometer measures its acceleration normalized



(a) Test set, with the null-class.



(b) Test set, without the null-class.

Figure 3.1: *t-SNE-projections of the Opportunity dataset with and without the null-class. The labels translate as 0 - stand, 1 - walk, 2 - sit, 3 - lie down, and 0 - null, 1 - stand, 2 - walk, 3 - sit, 4 - lie down, respectively.*

Table 3.1: *Number of samples for the datasets, all in 30 Hz.*

Dataset	OPP	OPP w/o null	KTH-KI-AA	KTH-KI-AA w/o transitions
# samples	676 713	559 928	676 800	576 000

to Earth gravity (g). The first day's recordings were of 5 adult subjects and the second day's recordings of 4 child subjects, two $\sim$ ten years old and two $\sim$ six years old.

The recordings of the group of adult subjects consisted of two "runs" of a predefined set of 14 categories, whereas the children on the second day only did one run through of 12 of the activity categories (vacuum cleaning and dishwashing were excluded). Two minutes were recorded per category per run, resulting in around 28 minutes of recordings per adult subject per complete run, and 24 minutes for the children's full run. See table 3.2 for the different activity sets. All data recordings were also filmed so that the starting and end times could be verified for the subsequent data labelling.

Activity categories

As can be seen in the list of Activity Set 1 (AS 1) in table 3.2, the 14 recorded categories are rather specific and fine-grained. The activities of category 12 and 13 (transitions between sitting, standing and laying down) were excluded after a number of trials because of their different (non-static) character compared to the other patterns: the laying down and sitting down could easily be confused with activities 0-4 with a data window size of only a second. This reduced the size of a run to 24 minutes for the adults, and 20 minutes for the children, resulting in all in all $24 \cdot 2 \cdot 5 + 20 \cdot 4 = 320$ minutes of relevant recordings (5.33 hours), or $320 \cdot 60 \cdot 30 = 576\,000$ data points. The null category, meaning the hours of recorded time surrounding the controlled activities was not included in the experiments on the KTH-KI-AA data.

In figure 3.3 are example raw data plots of segments from activity categories 8-11. Data from more categories can be found in appendix A.1.

After trials with AS 1-4, respectively, a few coarser category representations were chosen. This meant mapping all data from the narrower categories to two broader sets of categories; one with 5 labels and one with 4, the latter corresponding to the static activities in the Opportunity challenge. These two activity sets are also shown in table 3.2, as Activity Sets 5-6, where the mappings of the original categories to their new labels are displayed.

The mapping from specific to broader categories is done the same way as in the Opportunity dataset, where the (same) data was also labelled on two abstraction levels (corresponding to the gestures and static motion tasks).

Dimensionality and size of the dataset

The accelerometers used (ActiGraph GT3X) record acceleration in three dimensions each, at 30 Hz. Compared to the Opportunity dataset which has 113 features, the KTH-KI recordings are of a considerably lower dimension (6 features). In terms of number of samples they are however rather equal. Recall that Opportunity in total has 5.18 hours of non-null recordings, where the KTH-KI data has 5.33 hours.

Figure 3.3: *Plots of the raw data from categories 8-11. 20 second excerpts.*

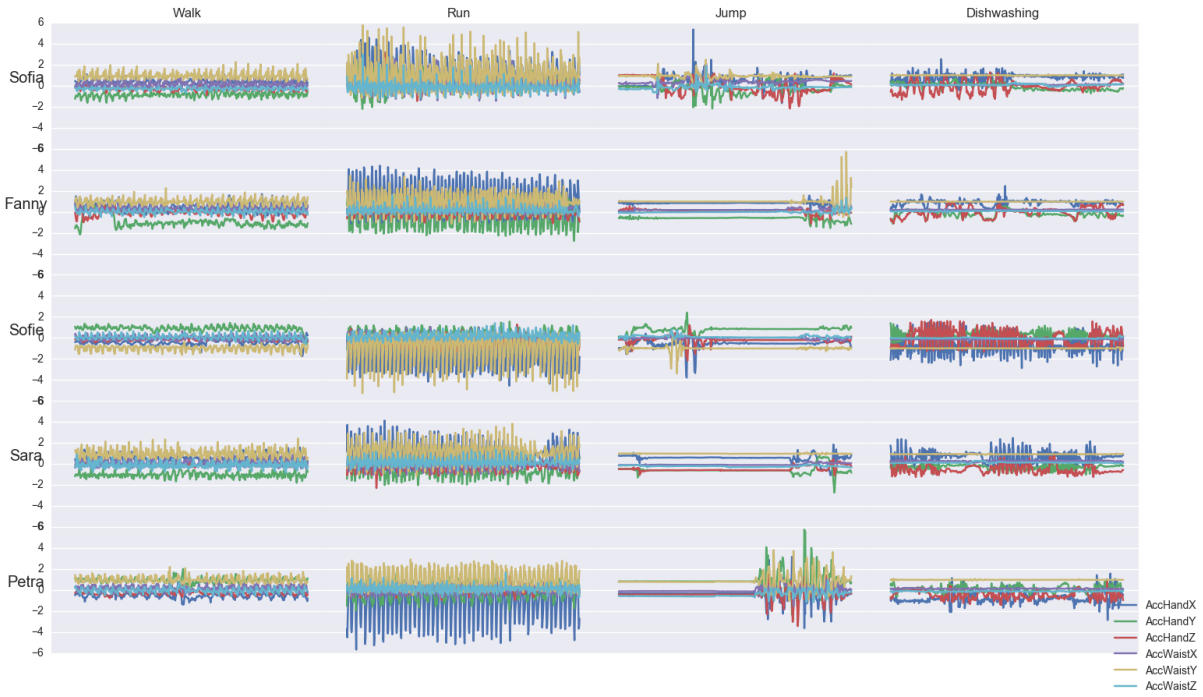


Table 3.2: *Activity sets for the KTH-KI AA dataset. AS 1-2 are for adult subjects and AS 3-4 for child subjects. Columns AS 5 and AS 6 show the two mappings of categories into coarser labels, for both populations.*

Label and category	**AS 1**	**AS 2**	**AS 3**	**AS 4**	**AS 5**	**AS 6**
0 Sit and read	x	x	x	x	2	2
1 Sit and draw	x	x	x	x	2	2
2 Sit and play on smart phone	x	x	x	x	2	2
3 Sit on floor and play	x	x	x	x	2	2
4 Lay down	x	x	x	x	3	3
5 Stand and draw	x	x	x	x	0	0
6 Get dressed	x	x	x	x	0	0
7 Vacuum	x	x	-	-	1	1
8 Walk	x	x	x	x	1	1
9 Run	x	x	x	x	1	4
10 Jump around	x	x	x	x	1	1
11 Dishwashing	x	x	-	-	0	0
12 10x "Sit-stand-sit"	x	-	x	-	-	-
13 10x "Lay down-stand-lay down"	x	-	x	-	-	-

Recordings outside of the laboratory

Another set of recordings were made, with 2 accelerometers worn on the hip and wrist by 4 adult subjects during two days in their respective everyday lives. There is data covering all the time during those two days that the subjects were awake and wore the sensors, around 28 hours per subject. These are not as carefully labeled, since the subjects themselves here reported per every half hour what level of activity intensity they had and what activity they engaged in and that was the only information I had to do the subsequent labelling.

The intention with these recordings was to see how well activity recognition can be performed on noisy data. My labelling of the data was done in accordance with the subjects reports, meaning that all the data points per half hour has the same label, which naturally cannot be in total correspondence with the accelerometer data.

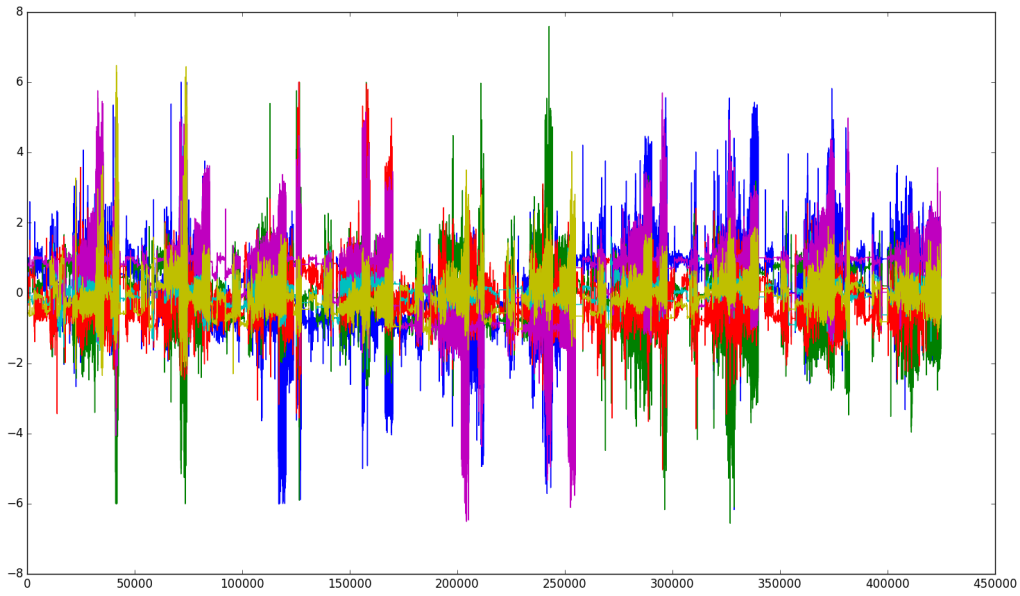
Training, validation and testing

The training and testing of the KTH-KI-AA dataset were performed on separate subjects. For the adult population, the models were trained on subjects 1-4 and tested on subject 5. For the child population (4 subjects), the training set consisted of three subjects and the test set of one subject. The test subjects were chosen blindly (randomly) when the dataset was saved. For the mixed population, the training set consisted of 8 subjects, and the testing set of one subject. Both a child and adult test subject were tested. For all experiments, a held out set of 20% of the training data was used as a validation set, for early stopping.

Pre-processing

No pre-processing was applied to the KTH-KI-AA data. In its raw form, the data is centered around 0 with a radius of around 6. See figure 3.4.

Figure 3.4: *The raw KTH-KI-AA feature channels in their entirety for the adult population.*



3.2 Human activity recognition – classification experiments

Supervised experiments on the data were performed with various architectures – three baseline models and two state-of-the-art architectures; the DeepConvLSTM by [6] and a bidirectional LSTM like in [5].

Additionally, as an introductory test, I trained one HMM per class for the 4-, 5- and 12-class data respectively and used them as classifiers by computing the probability for a test sequence given a model. This did not obtain better F1-scores than approximately 70% which illustrated the need for a richer model such as the neural networks described in sections (3.2.5-3.2.5).

3.2.1 Initialization

The default initalization in the Keras implementation of their Conv1D-, Dense- and LSTM layer is the Glorot uniform initialization, first introduced in by Xavier Glorot and Yoshua Bengio in 2010 in their paper *Understanding the difficulty of training deep feedforward networks* [38]. The biases are set to 0, and the weights are set according to the following uniform probability distribution:

$$W \sim U\Big[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\Big]. \tag{3.1}$$

3.2.2 Loss function

Categorical cross-entropy was used as loss function throughout the experiments. As touched upon in section 2.1, this is computed as

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}\big[\, y_{ij}\log(P_{ij})\,\big], \tag{3.2}$$

where the index i ranges over the training examples, and j ranges over the different possible classes, so that y_{ij} binarily represents whether a label is true for a particular training example.

3.2.3 Epochs and early stopping

The number of epochs for the training of models varied from around a hundred to five hundred epochs, depending on the training progress. Early stopping was employed in order to save computational resources. Early stopping is a form of both optimization and regularization (depending on how you see it), which ends the training process if the performance on the validation set has not improved for a certain number of epochs. The number of epochs of patience, as well as the maximum number of epochs, can be considered hyperparameters of the model. These are listed in appendix A.2.1.

3.2.4 Practicalities

Sliding window input

The input to the neural networks were sequences cut out from the data by a sliding window mechanism, across the temporal dimension (that is to say, not across the feature dimensions). The size, s_w, of these sliding windows was a hyperparameter during the experiments but the

step size when doing the segmentation was always half of the window size ($s_{step} = s_w/2$). If the number of feature channels for the data used was d, one sliding window would have the dimension $s_w \times d$. The use of a constant ratio of 2 between step size and window size can be found in both [6] and [5].

[6] uses a window size of 24 time steps and [5] one of 15 time steps, corresponding to $4/5$ and $1/2$ of a second, respectively, for 30 Hz data (which they, again, both use). (Semi-exhaustive) parameter grid search made me use a window size of 30 time steps for most of the models, corresponding to windows of one second's length, though for the baseline LSTM model I used $s_w = 60$. See appendix A.2.1 for more details on model parameters.

Labelling of a sliding window

The label for a sliding window was chosen as the majority category among the s_w data points of the segment.

The null category

The null category was excluded in all KTH-KI activity sets despite the unnatural transitions this creates in the sequential data, when from one datapoint to another the "scene" suddenly jumps. This means that the recordings between the pre-defined activities was removed from the dataset. These jumps were considered negligable, partly since both [6] and [5] obtained better results with their null categories removed than with them present.

3.2.5 Models

Stateful in Keras

When using the Keras implementation of LSTM layers, one can set the attribute *stateful* to either true or false. This decides whether the LSTM units should remember their states across batches, beyond the context of the separate sequences (which in my case often are short - one or two seconds). The last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch, according to the Keras [34] documentation. The baseline LSTM in this report uses that property, but not the DeepConvLSTM or the bidirectional LSTM, simply based on what performed the best.

Baseline models

The following baseline models were implemented and tested with different parameters (specified in appendix A.2.1) on both datasets:

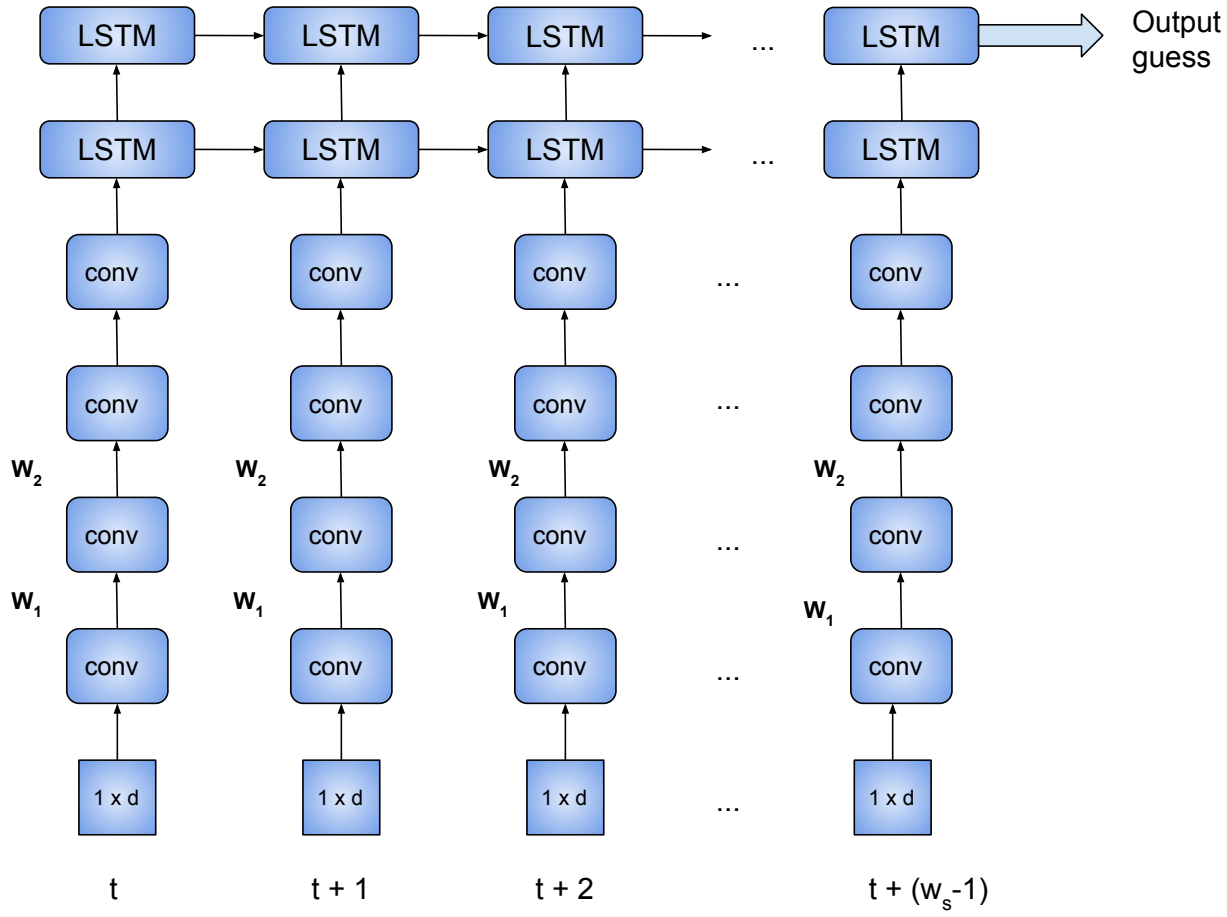
- **Baseline MLP** consisting of two hidden fully-connected layers both with $tanh$ activation, and a dense output layer with softmax activation
- **Baseline Convolutional Network** consisting of the following layers in order: 1D convolutional layer (64 filters with size 5) with ReLU activation, 1D max-pooling layer and a dense output layer with softmax output activation.
- **Baseline LSTM network** consisting of one hidden LSTM layer and one dense output layer with softmax activation.

They were all optimized using the Adam algorithm (see section A.4.1) for the reason that Adam is a widely used optimization algorithm for different applications of deep learning. [1]

DeepConvLSTM

The DeepConvLSTM consists of 4 convolutional layers (64 filters each with size 5) with $ReLU$ activations stacked on top of each other to extract features, followed by two LSTM layers with $tanh$ activations that are intended to handle the temporal sequentiality of the data. In both of the LSTM layers there is dropout regularization applied with 50% chance, in accordance with [6]. Lastly the model has a dense output layer with a softmax activation. The model is illustrated temporally unrolled in figure 3.5. The model was optimized with both RMSProp and Adam, to see if the optimization method made any difference. The reason that RMSProp was chosen was because that's the optimization method employed in [6], and the reason that Adam was chosen was, as for the baseline models, simply that it's a popular optimizer that works well for a variety of purposes [1].

Figure 3.5: *Schematic of the temporally unfolded DeepConvLSTM. Before the actual output guess, the data passes by a dense output layer with softmax activation. The W_i are short for the different weight matrices present in one LSTM layer, see section 2.2.4.*



Bidirectional LSTM

A network like the bidirectional LSTM network in [5] was implemented. Such an architecture was first introduced by Alex Graves and Jürgen Schmidhuber in 2005 [14]. At the time of writing this report, this architecture holds the state of the art score on the gestures task of the Opportunity challenge. It consists of two LSTM layers, one of them processing the input sequences in the forward direction, and one of them in the backward direction. Consequently, when processing any point in the sequence, network has already taken in all context before and after that point. They are then merged by the sum of their respective elements, and at the end they are joined in a dense output layer with softmax activation.

Bidirectional architectures are appropriate when one is dealing with offline data, meaning that the dataset in its entirety is available and one can extract patterns from its full synthesis. For natural reasons, if the data is online and arriving in a streamed format, one can not start at the "end" of the data (the future data) and go backwards.

The model was optimized using both Adagrad and Adam, for similar reasons as the optimizers were chosen for the DeepConvLSTM. [5] used Adagrad in their paper, but I also wanted to try Adam to see if there was a difference.

3.2.6 List of experiments

See table 3.2 for a specification of the activity sets.

---

1. **Opportunity**
   Reimplementation of the DeepConvLSTM from [6] and the bidirectional LSTM like in [5], as well as trials with other models.
   (a) Classification of static/periodic movement (5 classes)
   (b) Classification of static/periodic movement without the null class (4 classes)
2. **KTH-KI-AA, adult subjects**
   (a) Classification of AS 2 (12 classes)
   (b) Classification of AS 5 (4 classes)
   (c) Classification of AS 6 (5 classes)
3. **KTH-KI-AA, child subjects**
   (a) Classification of AS 2 (12 classes)
   (b) Classification of AS 5 (4 classes)
   (c) Classification of AS 6 (5 classes)
4. **KTH-KI-AA, mixed population (7 and 9 subjects)**
   The 7 subject population consists of all 5 adult subjects and the two older children, and the 9 subject population consists of all 5 adults and 4 children.
   (a) Classification of AS 2 (12 classes)
   (b) Classification of AS 5 (4 classes)

   (c) Classification of AS 6 (5 classes)

5. **KTH-KI-AA, self-reported data for 4 subjects**
   Train on laboratory data, test on self-reported data.

   (a) Classification of AS 5 (4 classes)

   (b) Classification of AS 6 (5 classes)

6. **Train on KTH-KI-AA-4/5, test on Opportunity (but only hip and right wrist-accelerometers)**
   The reason I only trained on the 4- and 5-class data of KTH-KI-AA data is because the Opportunity data classes would not match the 12-class data. The Opportunity classes are walk, stand, sit and lie down which are trained for both in KTH-KI-AA-4 and 5. It should be noted here that the feature channels of the two datasets live in different intervals, the Opportunity data is pre-processed to the interval $[0, 1]$ whereas the raw KTH-KI-AA data is approximately in the interval $[-7, 7]$ (see figure 3.4). I mapped the Opportunity data to the interval $[-7, 7]$ for this experiment.

   (a) Classification of AS 5 (4 classes) with all data in the (approximate) interval $[-7, 7]$.

   (b) Classification of AS 6 (5 classes) with all data in the (approximate) interval $[-7, 7]$.

Chapter 4

Results and discussion

This section will list the results from the different experiments. More emphasis will be put on the experiments 1-2, since this data is cleaner than for the other populations and gave more stable results. At the end of the section there will be some discussion about the datasets followed by a summary of the most important results and a general discussion.

4.1 Evaluation metric

The metric of a model's performance employed in this study is the weighted F1-score. Regular F1-score is computed as follows:

$$F_1 = 2\frac{pr}{p+r}, \tag{4.1}$$

where p and r are, respectively, precision and recall. Thus, the F1-score is the harmonic mean of those two quantities.

Precision is the fraction of accurate classifications out of the total classifications made for one class, and recall is how many accurate classifications were made out of all the samples from that class present in the dataset at hand.

More concisely, if TP is true positive, FP false positive and FN is false negative, then precision is $\frac{TP}{TP+FP}$ and recall is $\frac{TP}{TP+FN}$, per class. The weighted F1-score is meant to counter possible class imbalance in multi-class classification:

$$F_{1,\text{weighted}} = \sum_i 2w_i \frac{p_i r_i}{p_i + r_i}, \tag{4.2}$$

where $w_i = n_i/N$, n_i is the number of samples of category i in the data and N is the total number of samples.

4.2 Results tables for experiments 1-6

The results in the tables are the weighted F1-scores obtained on the test sets using the model that performed best on the validation sets during training.

4.2.1 Experiments 1-2: Opportunity dataset and KTH-KI-AA adult population

Table 4.1 covers experiments 1 and 2, meaning the experiments on Opportunity and on the adult population of KTH-KI-AA. The baseline LSTM is the model that performs best on

KTH-KI-AA, whereas the bidirectional LSTM performs best on Opportunity. Next after the baseline LSTM comes the baseline CNN, at 83.87% F1-score on KTH-KI-AA-4 (compare to 77.33% with the bidirectional LSTM and 75.64% with the DCLSTM).

It's interesting to note that the baseline CNN has only 2244 parameters compared to the 138 772 parameters of the bidirectional LSTM (4-class task) (see appendix A.2.1). It seems that the complexity and number of units is not decisive for the KTH-KI-AA data. Although the best model, the baseline LSTM, has more parameters than both of the just mentioned ones (166 404 parameters), the DCLSTM with its 294 532 parameters scores way lower than the baseline CNN. Something else is at play, and the important thing seems to be to capture any spatial or temporal structure in the data rather than more detailed fine-grained structures made possible to detect with more hidden units and layers. In fact, there might not be all that much fine grained patterns to pick up from samples such as these of just $s_w * 6$ dimensions.

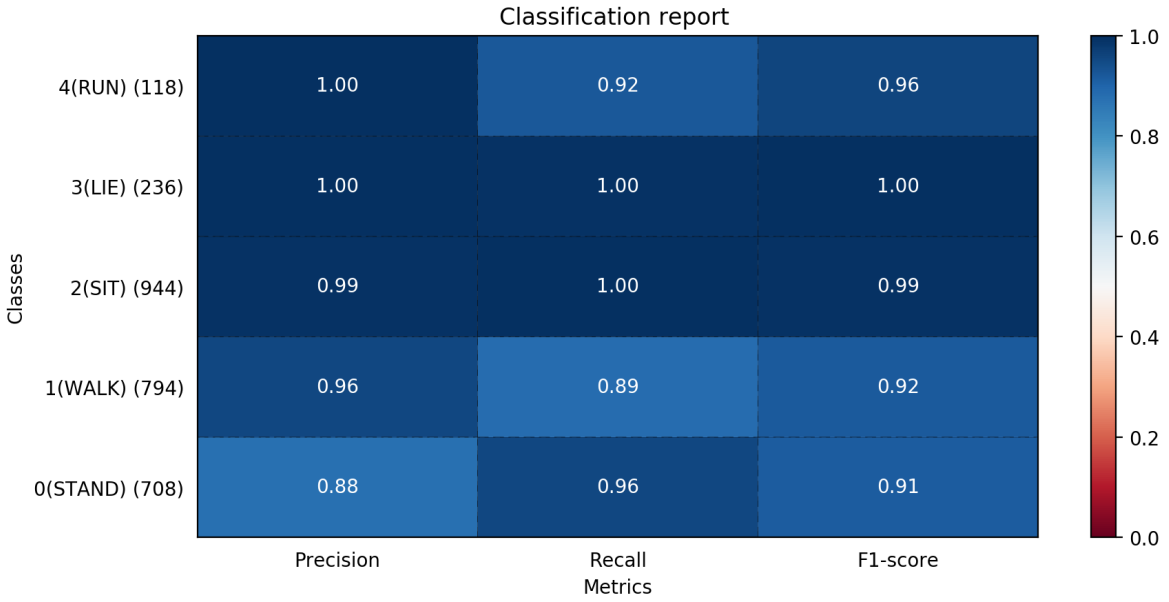
Another general trend to observe is that the DeepConvLSTM and the bidirectional LSTM do not perform so well on KTH-KI-AA, whereas they have high accuracies for Opportunity. And the inverse relation holds as well, the baseline models which are the best on the KTH-KI-AA data don't obtain as high scores as the two state-of-the-art models on Opportunity. This gives us a strong hint that more complex models are more suited for datasets of higher dimension, and simpler models perform better on lower dimensional datasets.

[6] obtained slightly better results on the Opportunity data with DeepConvLSTM, which difference might be due to random initializations of the network. My runs with Glorot uniform initalization can sometimes vary with as much as ± 5 percentage units in the F1-score (the best runs for the different models are shown here).

As the table shows, the classification of KTH-KI-AA-12 (activity set 2 with 12 classes) did not work well. The results are often barely above the performance of a random classifier (which should perform around $\frac{1}{n_{classes}} = \frac{1}{12} \approx 8.3$).

The 12-class (AS 2) labelling of the KTH-KI-AA test set is shown as a t-SNE projection in figure 4.4, where one can observe that the different categories are not very separated. This tells us that the information from two accelerometers worn on waist and wrist are perhaps not enough to distinguish between two hand-dominated activities such as for instance categories 0 and 2 in AS 2; sitting and reading and sitting and playing on a screen device.

In figure 4.2, class reports are shown for the experiments on the adult population for KTH-KI-AA-4/5 performed with the models DeepConvLSTM and the bidirectional LSTM. Two trends that can be observed across these models is that the category 'lie down' seems to be the easiest to distinguish, and that the category 'stand' on the contrary seems to be the most difficult to classify. In comparison, the baseline LSTM model performs more evenly across all the different categories (see figure 4.1a) than DeepConvLSTM and the bidirectional LSTM. This is presumably due to the fact that the baseline LSTM takes in a larger context in the classification, it looks at the longer time dependencies, and doesn't make its decision from just one pattern in a 1-second window. The ability to perform more evenly across categories is furthermore illustrated in figure A.3 in the appendix, where the 12-class results are presented for the baseline LSTM and for the DeepConvLSTM (second best model on the task). In the figure, we observe how the DeepConvLSTM (although with a worse score overall) has 0.0% F1-score for several categories but around 70% for a few.

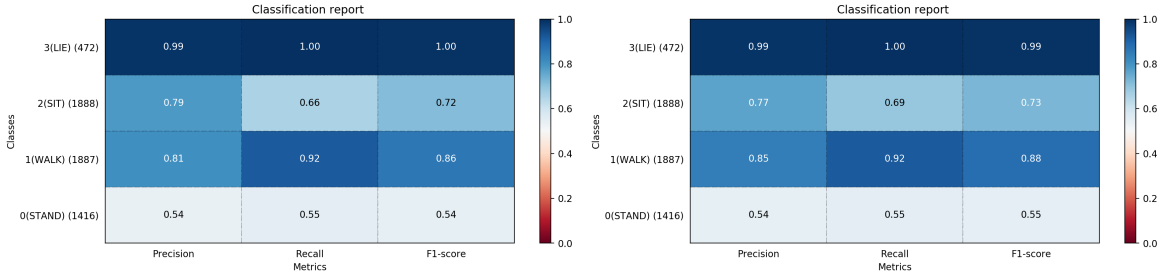


(a)

Actual \ Predicted	Stand	Walk	Sit	Lie down	Run
Stand	677	25	6	0	0
Walk	90	703	1	0	0
Sit	3	0	941	0	0
Lie Down	1	0	0	235	0
Run	1	8	0	0	109

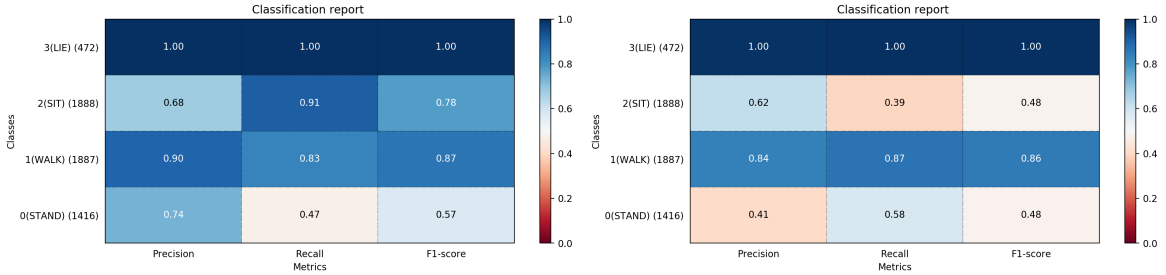
(b) Confusion matrix

Figure 4.1: *The precision, recall and F1-scores in 4.1a as well as the confusion matrix in 4.1b for the best performing model on the KTH-KI-AA dataset; the baseline LSTM. Shown for the 5-class task. The numbers to the right of the category names in 4.1a are the number of samples per class in the test set.*



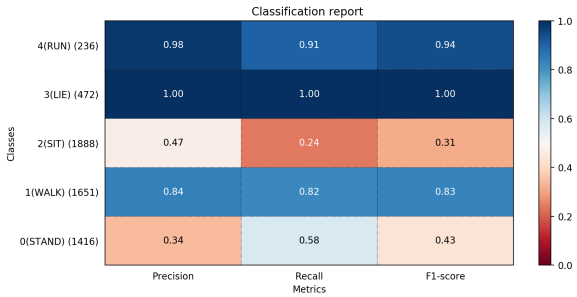
(a) DCLSTM, RMSProp

(b) DCLSTM, Adam

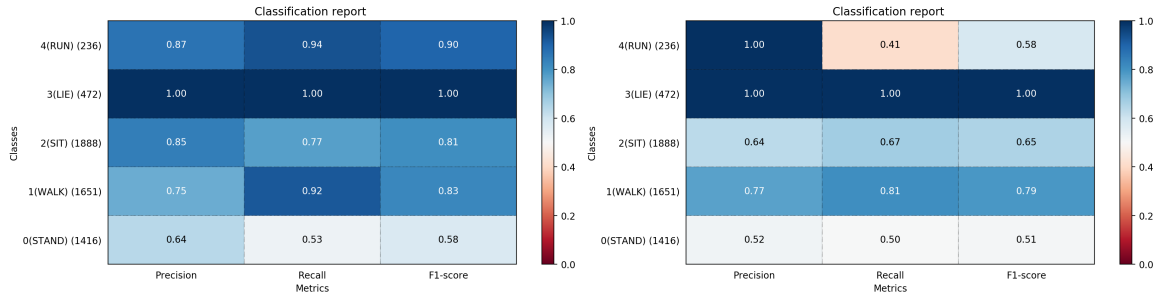


(c) B-LSTM, Adam

(d) B-LSTM, Adagrad

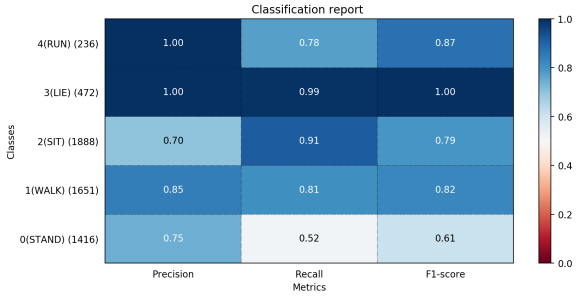


(e) DCLSTM, RMSProp



(f) DCLSTM, Adam

(g) B-LSTM, Adagrad



(h) B-LSTM, Adam

Figure 4.2: *Precision, recall and F1-score per class for the classification with DeepConvLSTM and B-LSTM. Classification of 4 and 5 classes respectively on the KTH-KI-AA dataset's adult population. The numbers to the right of the class names show how many samples there are from each class in the test set. Note how the category 'lie down' seems to be the easiest to classify, and how 'stand' seems to be the most difficult. Some corresponding confusion matrices are shown in figure 4.3.*

Table 4.1: *Weighted F1-scores for the experiments 1 a-b and 2 a-c (Opportunity and KTH-KI adult subjects).*

Model	Optimizer	OPP	OPP w/o null	KTH-KI-4	KTH-KI-5	KTH-KI-12
HMM (8 states)	n/a	-	-	0.6095	0.7071	0.2253
Baseline FC	Adam	0.7821	0.8278	0.6245	0.6192	0.1040
Baseline CNN	Adam	0.7947	0.8617	0.8387	0.8053	0.0802
Baseline LSTM	Adam	0.6168	0.8482	**0.9500**	**0.9519**	**0.3897**
B-LSTM	Adam	0.8426	0.9183	0.7733	0.7424	0.1868
B-LSTM	Adagrad	**0.8541**	**0.9190**	0.7131	0.6925	0.0918
DCLSTM	Adam	0.8145	0.9084	0.7564	0.7758	0.2002
DCLSTM	RMSProp	0.8330	0.9096	0.7449	0.5774	0.1585
DCLSTM by [6]	RMSProp	0.895	0.930	-	-	-

Confusion matrices

Some confusion matrices from the runs with DeepConvLSTM and the bidirectional LSTM, whose results are in figure 4.2, are shown in figure 4.3.

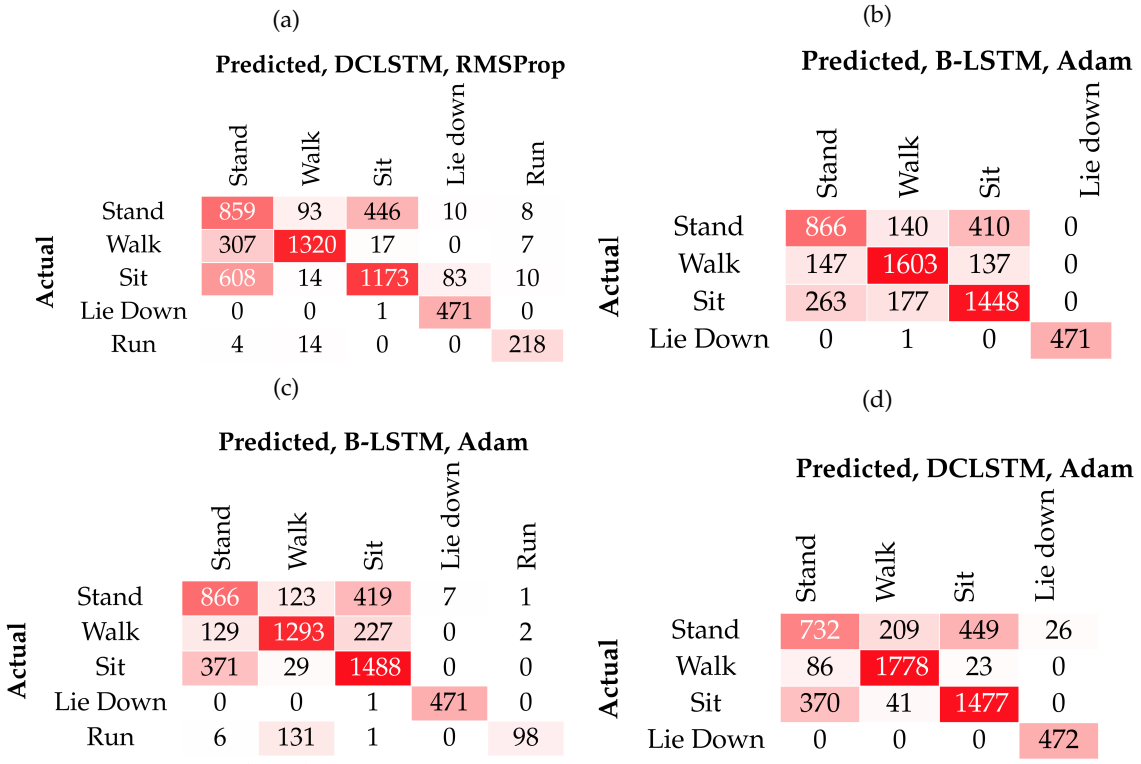


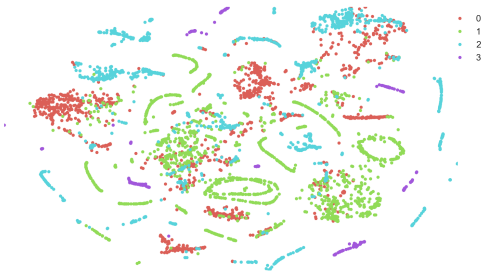
Figure 4.3: *Confusion matrices for DeepConvLSTM and the bidirectional LSTM.*

As can be seen from all 4 of the matrices in figure 4.3, the *standing* (0) and *sitting* (2) categories are often mixed up. This is partly explained by studying the t-SNE-projections of the KTH-KI-AA-12 (AS 2) dataset in figure 4.4, where one can see for example in the top right cluster that category 11, *dishwashing*, and category 1, *drawing sitting*, are mixed up. A reason for this could be that they are both activities where the hands are used actively. Since *dishwashing* is mapped to standing and *drawing sitting* to sitting in the coarser labelling, these two are still confused for the 4- and 5-class tasks as is visible in their corresponding top right clusters (figures 4.4a-4.4b). Category 3, *sitting on the floor and playing*, is also somewhat mixed up with the mentioned hand-dominated activities. This particular activity was recorded rather freely but many of us were playing with some kind of pen or object with our hands which could provide an explanation for this.

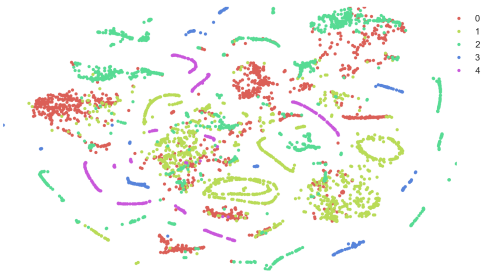
Continuing the study of KTH-KI-AA-12 in figure 4.4c, it seems that the data points for *vacuum cleaning* and *dishwashing* are also similar, along with *dressing* (see the bottom right – blue and fuchsia). These are also activities where the hands are active, but both are standing, which would explain why their cluster is separate from the *dishwashing-drawing* cluster. *Vacuum cleaning* was, perhaps arbitrarily, mapped to the walking class in the coarser labelling. This could be another plausible reason to the confusion in the classification between walking and standing (in the 4- and 5-class task).

Figure 4.5 shows the KTH-KI-AA-5 data after it has gone through a convolutional- and an LSTM layer, respectively. One can note that especially the LSTM layer output shows more separate class clusters than for the raw data.

(a) 4 categories



(b) 5 categories



(c) 12 categories

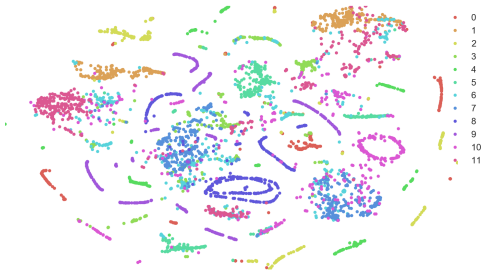
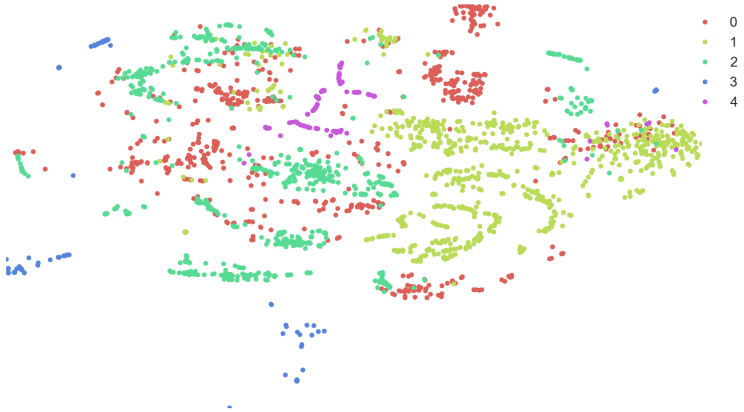
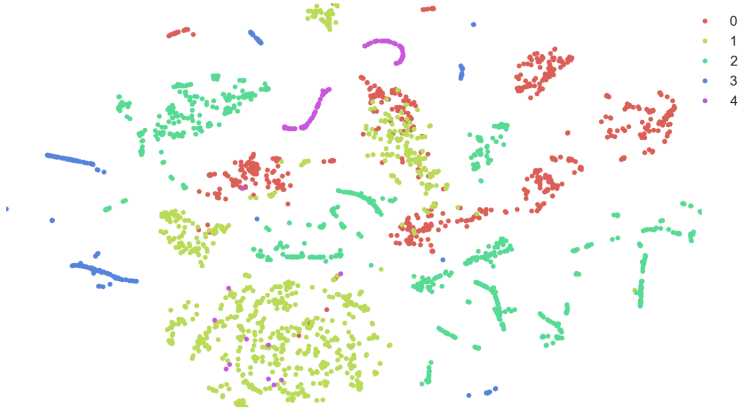


Figure 4.4: *t-SNE-projections of the test sets of the KTH-KI-AA datasets for 4, 5 and 12 categories. The labels in figure 4.4a and 4.4b translate as 0 - stand, 1 - walk, 2 - sit, 3 - lie down and 4 - run. The labels in figure 4.4c translate as 0 - sit and read, 1 - sit and draw, 2 - sit and play with a screen device, 3 - sit and play on the floor, 4 - lay down, 5 - stand and draw on a board, 6 - get dressed, 7 - vacuum clean, 8 - walk, 9 - run, 10 - jump, 11 - wash dishes.*



(d) Output from the convolutional layer from the baseline CNN.



(e) Output from the LSTM layer from the baseline LSTM.

Figure 4.5: *Feature extracted data. t-SNE-projections of intermediate layer outputs of the KTH-KI-AA-5 dataset. The labels translate as 0 - stand, 1 - walk, 2 - sit, 3 - lie down, 4 - run. Note that the classes are slightly more separated than in figure 4.4b.*

Table 4.2: *F1-scores for the experiments 3 a-c (KTH-KI child subjects).*

Model	Optimizer	KTH-KI-4	KTH-KI-5	KTH-KI-12
Baseline FC	Adam	0.2691	0.4142	0.0534
Baseline LSTM	Adam	0.4220	**0.5130**	**0.0722**
Baseline CNN	Adam	0.4724	0.4838	0.0658
B-LSTM	Adam	0.4387	0.4097	0.0526
B-LSTM	Adagrad	0.5002	0.4012	0.0449
DCLSTM	Adam	0.4960	**0.5171**	0.0607
DCLSTM	RMSProp	**0.5380**	0.4463	0.0561

4.2.2 Experiment 3: Child population

Table 4.2 shows the classification results on the child population of the KTH-KI-AA dataset (experiments 3 (a-c)). These results are considerably worse than the ones for the adult population. This may be due to larger individual variations among the child subjects (related to low motivation or understanding for proper data collection, or to the varying ages among the subjects - 10, 9, 7 and 6 years old). It's also unclear which subject constituted the test set here. It may be that it was one of the younger children, whose data was less clean than the older ones. The data was in general less clean for the children than for the adults, in any case. The DeepConvLSTM and the baseline LSTM performed the best on this data (in the 5-class problem they were only 0.4% apart from each other). The results for the baseline CNN and the bidirectional LSTM were comparable for the 4-class task but the CNN was better at the 5-class task.

4.2.3 Experiment 4: Mixed population

The two models that performed the best on KTH-KI-AA; the baseline CNN and the baseline LSTM were used to perform experiments 4 (a-c), which were made on the total, mixed population of the KTH-KI-AA data. These results are shown in table 4.3 and 4.4. They are again less accurate than for exclusively the adult population on the 4- and 5-class tasks, although they are better than the children-only results. Notably, however, they do present the best result for the 12-class task with 55.33% F1-score. See figure 4.6. This gives us a hint that if we want to be able to learn more specific activities with simple models such as the baseline models of this report, we might benefit from collecting more, and more diverse, data.

One might have expected worse results for the mixed population than for the child population, since the inter-individual variations are even larger in a mixed population dataset. In the mix, you have both the noise from the child data (due to children being children and their lack of motivation to perform activities in a concentrated way) and the variation between adults and children. However, on the 4-class task for 9 subjects (train on 8 subjects, test on 1), when the test subject was an adult, 89.55% was obtained – a drastic improvement compared to the best result on the 4-class task for the child population (53.80%).

The most probable reason for this seems to be that there is more training data for the mixed population case than for the child population case. One thing that supports that is the visible steady increase in accuracy between the 7 subjects and 9 subjects experiments in tables 4.3 and 4.4, across the 4-. 5-. and 12-class tasks, speaking in favor of adding more data to the training.. Though, there is instability between the adult and child test subjects for the

Table 4.3: *F1-scores for the experiments 4 a-c (KTH-KI mixed population, 7 subjects).*

Model	Optimizer	Test subject	KTH-KI-4	KTH-KI-5	KTH-KI-12
Baseline CNN	Adam	Adult	**0.6632**	0.3554	0.1889
Baseline CNN	Adam	Child	0.6331	0.4971	0.3574
Baseline LSTM	Adam	Adult	0.6128	0.3911	**0.3607**
Baseline LSTM	Adam	Child	0.6195	**0.5823**	0.1592

Table 4.4: *F1-scores for the experiments 4 a-c (KTH-KI mixed population, 9 subjects).*

Model	Optimizer	Test subject	KTH-KI-4	KTH-KI-5	KTH-KI-12
Baseline CNN	Adam	Adult	0 .7299	**0.7136**	0.4757
Baseline CNN	Adam	Child	0.5291	0.5255	0.2861
Baseline LSTM	Adam	Adult	**0.8955**	0.5668	**0.5533**
Baseline LSTM	Adam	Child	0.3784	0.5969	0.3399

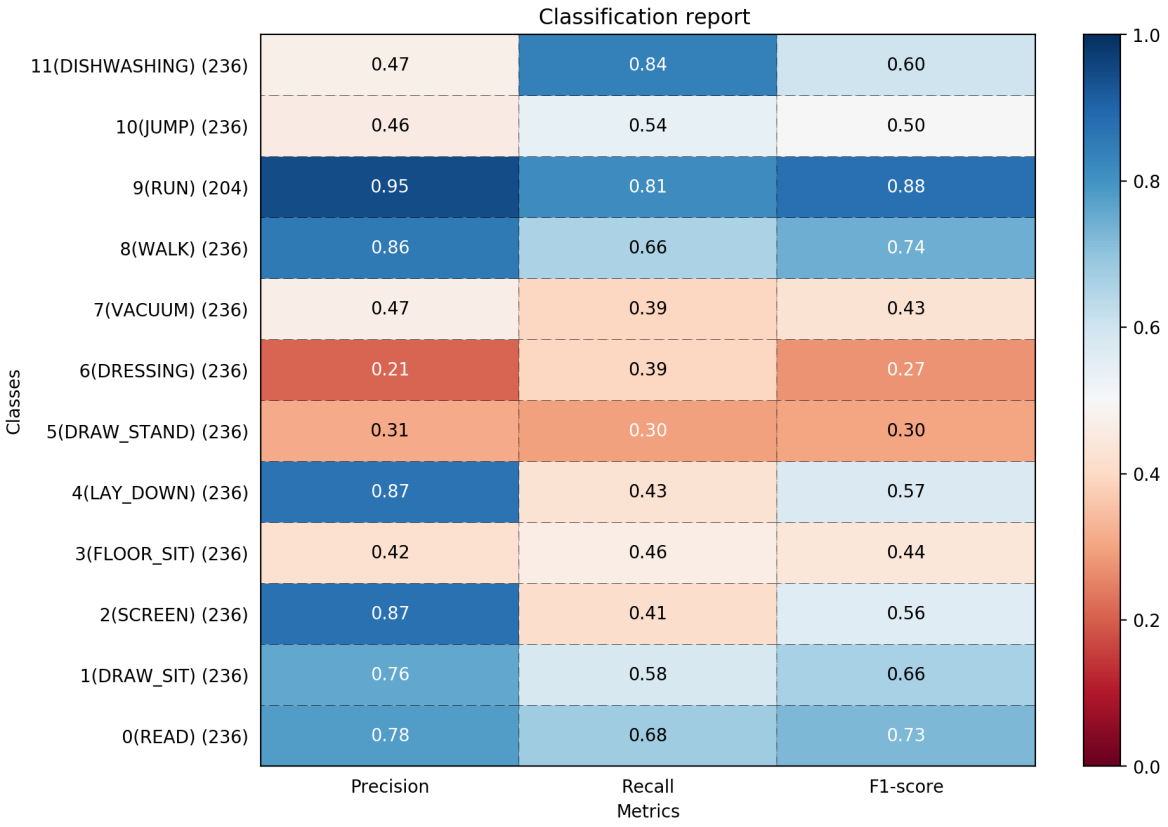
9 subject experiment, compared to the 7 subject experiment. This is likely explained by the fact that in the 7 subject experiment the only possible child test subjects were the older and more "well behaved" ones.

4.2.4 Experiment 5: Self-reported data

The intention with this experiment was to see if a model trained by the laboratory data could generalize to data recorded in more natural environments. The resulting scores are lower than in experiments 2 and 4 but this was to be expected given the many erroneous labels that had to be set due to the label source of information being a bi-hourly report over a subject's activities during a day.

The self-reported data was recorded two times per hour by the subject herself, thus making its labelling intrinsically noisy. If a subject sat down and had a meal for two thirds of one half hour, and stood up for the last ten minutes, that half hour is still labeled as sitting if the subject marked it as "sat down to eat". Or – due to human error and the fact that it's hard to remember exactly when and for how long one did what, it might as well have been labelled as "standing".

Because of that, the results in table 4.5 don't reflect the true classification capacity of the models. Rather, it says something about how poor interpolating questionnaires can be in reflecting true, second-by-second activity, which referring back to section 1.1 again underlines the purpose of the thesis. When it comes to which model performed the best for the self-reported data, there is no clear trend. Naturally, the results variation among the subjects



(a)

Predicted, DCLSTM, RMSProp

Actual	Read	Draw, sit	Smartphone	Play on floor	Lay down	Draw, stand	Dress	Vacuum	Walk	Run	Jump	Dishwash
Read	160	12	0	41	2	19	1	1	0	0	0	0
Draw, sit	1	138	4	0	0	28	41	0	0	0	15	9
Smartphone	12	0	97	1	0	81	42	0	0	0	2	1
Play on floor	0	7	0	109	2	0	89	25	0	0	0	4
Lay Down	0	5	8	27	101	26	42	9	0	1	17	0
Draw, stand	0	2	1	38	6	70	78	12	0	0	0	29
Dress	0	4	0	19	4	0	93	52	0	0	6	58
Vacuum	0	1	0	16	0	0	45	93	0	0	13	68
Walk	0	0	1	0	0	0	11	0	155	2	57	10
Run	0	1	0	0	0	1	2	0	3	166	28	3
Jump	16	4	0	9	1	1	0	6	23	6	128	42
Dishwash	16	8	0	0	0	0	0	1	0	0	12	199

(b) Confusion matrix

Figure 4.6: *The precision, recall and F1-scores in 4.6a as well as the confusion matrix in 4.6b for the best performing model on the 12-class task for KTH-KI-AA; the baseline LSTM trained on the full, mixed population (see table 4.4). The numbers to the right of the category names in 4.1a are the number of samples per class in the test set.*

Table 4.5: *F1-scores for the experiments 5 a-b (KTH-KI-AA, self-reported data).*

Model	Optimizer	Subject 1	Subject 2	Subject 3	Subject 4
Baseline FC	Adam	0.2689	0.4298	0.3062	0.4187
Baseline CNN	Adam	0.3247	0.4632	0.3400	**0.5482**
Baseline LSTM	Adam	0.3611	**0.5439**	0.3699	0.4585
B-LSTM	Adam	0.4626	0.3523	0.3125	0.3943
B-LSTM	Adagrad	0.4549	0.4458	0.3488	0.4118
DCLSTM	Adam	0.4898	0.4412	**0.3858**	0.4659
DCLSTM	RMSProp	**0.5130**	0.3667	0.2780	

could be due to differences in accuracy while performing the self-report.

A rough estimation taking into account the factors mentioned above might give that half of the labels are false in self-reported data of this kind. With that in mind, results of around 54% F1-score seems at least promising for future trials with "real life" data.

4.2.5 Experiment 6: Train on KTH-KI-AA-4/5 and test on Opportunity (and vice versa!)

Results in table 4.6. Here, I was hoping to be able to generalize from the KTH-KI-AA dataset to another population. The result was not satisfying, but there can be different reasons for this.

First of all, it turned out that 7% of the data from the hip and right wrist-accelerometers from the Opportunity dataset were "NaNs". By default while training on the Opportunity dataset for the other experiments I had set these to 0 (overall there were 4% NaN in the entire dataset). Of course, this introduces artefacts into the data. For this experiment, I also tried to just remove any datapoint that contained a "NaN" in any of its feature columns. This increased the score somewhat. Of course, again, this introduces artefacts and in particular perturbs the sequentiality since there will be many jumps introduced. The results below are all obtained when the Opportunity data has had the NaNs removed, which in the end was what gave me the best results. The score after training on KTH-KI-AA-4 is just below random performance while the score after training on KTH-KI-AA-5 is 9.4 percentage points above random performance.

Another important reason that the features didn't transfer over to Opportunity could be that the placement of and the equipment itself were different for the two datasets. Even though the features I took from Opportunity to compare were accelerometers placed on hip and wrist, there exist different makes of accelerometers, and [6] does not reveal which model of accelerometer they used.

Moreover, at the Opportunity data recording sessions, the subjects were wearing sensors on almost the whole body. This might have influenced their movement patterns. Also, all their recordings were in a kitchen, and the predefined tasks were related to a kitchen setting, which is another thing different from the KTH-KI-AA recordings. The latter were made in other types of rooms (conference room, large university building hallway, corridors), and the activities were more general.

Table 4.6: *Results when transferring knowledge across the two datasets. Results are shown for the best performing model.*

Train	Test	F1-score	Model
KTH-KI-AA-4	Opportunity	0.2431	Baseline LSTM
KTH-KI-AA-5	Opportunity	0.2940	Baseline LSTM
Opportunity	KTH-KI-AA-4	0.4108	DCLSTM

4.3 KTH-KI-AA Dataset

The activity categories in the KTH-KI-AA dataset are somewhat arbitrary. This is because those of us involved in the project wanted a broad spectrum of activities that were typical both for adults and children. The project has to be regarded as a pilot study, where we wanted to ascertain feasability of these kinds of machine learning methods on accelerometer data. It can thus be seen as the first step toward a possible larger study where the activity categories can be chosen with more thought behind them. The goal is as previously stated to at some point be able to measure activity objectively from data caught in the wild.

As has already been touched upon, two of the children subjects did not always conform to the listed activities. Notably, during the recording of walking they tended to start running instead, and they could not lie entirely still during the lying down sessions. There is thus noise in the labelling of their data.

Subject 0 kept track of the time while recording the activities. This means that subject 0's activities were not always performed in the most natural way, since the subject had to sometimes look at her clock. In the beginning and end of the activities, this was taken into account. If the measurement of one activity started at 11.37.00, the labelling of the activity doesn't start until two seconds later. This is both to take into account the time it took for subject 0 to start the activity, but also for the rest of the subjects' reaction time.

Lastly, the adult subjects were all women aged between 23 and 30. A larger study would benefit from having more varied test subjects.

4.4 Summary and general discussion

The best obtained results on the KI-KTH-AA dataset were a weighted F1-score of 95.0% on the 4-class task (experiment 2 (b)), and 95.2% on the 5-class task (experiment 2 (c)). This is significantly higher than Ordoñez *et al.*'s [6] results on the Opportunity 5-class task, and close to the 96% accuracy of Saez *et al.* [11] on a different (also more high-dimensional and multi-modal) dataset.

Additionally, [6] report a 68.9% weighted F1-score when performing the gestures (18 class) task with features only from accelerometers, like in our collected data. The difference is that they use 5 accelerometers (15 sensor channels) where we use two (6 sensor channels). This is only tested for the gestures data and not for the 5-class static/periodic motion data.

Certainly, the 18-class gestures task of the Opportunity challenge is not the same as the 5-class task, and might at first seem more challenging than a 5-class task. However, [6] actually report a better result on the gestures task than on the 5-class task. This leads me to believe that they might obtain a result comparable to and not necessarily better than 69% restricted to only accelerometers on the 5-class task with DeepConvLSTM. With that in mind,

and the authors of [6] being state of the art researchers, and measurements from two accelerometers being a lot less information than the measurements from 5 accelerometers, the results of 95.0% and 95.2% weighted F1-scores on the KTH-KI-AA dataset can be considered successful.

The results are especially promising seen that the parameter search performed for the different architectures can only be considered semi-exhaustive. There might be more optimal parameter settings for the architectures than have been laid out here to be found in a more rigorous parameter search, for example using a framework like the *fANOVA* analysis framework [39]. The training of most of the networks could take hours, even using GPUs, which time limited my possibilities of testing every idea.

Furthermore, with regard to the unreliability of self-reported questionnaires about daily activities of subjects, being able to measure activities that are to 95% accurately mirroring the performed activities can be considered an important step forward for the Karolinska Institute's Department for Public Health in measuring human activity. Specifically, the two categories that are confused most often in my best model's classification are standing and walking. This means that if one is especially interested in separating sedentary from non-sedentary behavior, which has been expressed from the Karolinska Institute, this can be done with even higher accuracy than 95%.

The collection of data from children subjects was not easy, and the classification results from that data should therefore not be interpreted as being rock solid. This might be good to know for future work, where one might want to consider letting the children do activities freely while filming them, and then label them afterwards, although this is labor intensive.

The training and testing data in the Opportunity challenge, respectively, contain data from some overlapping subjects. I considered it more clean to do classification on new test subjects, which is also the method of for example [11]. For any practical application, from the perspective of the Dept. for Public Health of the Karolinska Institute, this is something that needs to work. Training data from all potential test subjects cannot realistically be collected.

On the other hand, to the advantage of the Opportunity challenge is that their test set consists of only the more natural activities (ADL) where the subjects could choose freely from a list of acitivities to occupy themselves with. This is something that my training and test data lacks: activities that follow upon each other in a more natural order. My dataset is simply cut out recordings from activities that were imposed upon the test subjects during a fixed amount of time. This might be a problem for generalisability to wild data. Still, the error rate on the non-laboratory self-reported data of around 50% should mostly be due to the intrinsic noise in the labelling (what with the activities only being reported every half hour) and shows that there is in effect potential for generalisation to noisy data.

4.4.1 Deep or shallow?

One question has been looming in the background throughout the work with this thesis: how complex does a neural network have to be in relation to its dataset, and how deep does it have to be? The theory of deep learning is currently being developed and is not yet mature to provide general answers regarding for example the required depth for a given task and dataset. Throughout the field, both researchers and practitioners rely on heuristics shown to work well for specific instances of problems [1].

But there are some theoretical results – recently for example by Poggio *et al.* [40], [41]. They look at specific, idealized classes of functions of hierarchical and compositional func-

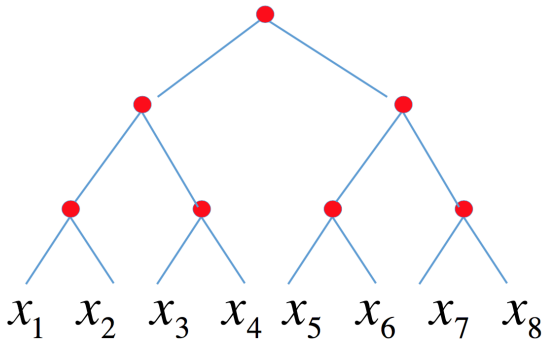
tions, of the following general form:

$$f(x_1, \ldots, x_n) = h_l \ldots (h_{21}(h_{11}(x_1, x_2), h_{12}(x_3, x_4)), \quad h_{22}(h_{13}(x_5, x_6), h_{14}(x_7, x_8) \ldots)) \ldots)). \tag{4.3}$$

This kind of function can be thought of as a binary tree. For a function of 8 variables, this is illustrated in figure 4.7 where the red dots are the different constituent functions h_i.

Their work contributes with bounds for what theoretical accuracy one can expect using a certain number of parameters, for shallow and deep networks respectively. In essence, this means that they can say when one should use a deep network and when not to, if the function we want to approximate is of the form shown in equation (4.3).

Figure 4.7: *Hierarchical, compositional functions can be represented as binary trees. The h_i-functions in equation 4.3 are represented by the red dots.*



The point made in [41] is that these types of hierarchical functions can be approximated arbitrary well by both shallow and deep networks, but that the number of parameters needed for a deep network to do the work is much smaller. The classical result [26] presents, as previously stated, no bound on the number of (hidden) units that are needed for a universal approximator of only one hidden layer.

The question then becomes, how do we know if we are in fact dealing with a compositional function in the case of mapping from low dimensional accelerometer data to activity categories? We naturally don't have access to the true closed-form function that handles that mapping. Until the proper theory is in place, we can try and rely on empirics. The deeper network DeepConvLSTM did not perform better on the KTH-KI-AA data than a baseline LSTM network: this is something we know. This leads us to conclude (although without being overly confident) that accelerometer data of low dimensionality doesn't have a very pronounced hierarchical form, but indeed a strong sequentiality. It is not surprising that data of low dimensionality is less hierarchically structured, since there simply is less information present there to gather structure from.

Chapter 5

Conclusions

Through the work with this thesis I have found that human activity recognition can be performed with near state-of-the-art accuracy using only two accelerometers when the activities to be classified are rather broad, like the set of 5 categories walk, stand, sit, lie down and run. To distinguish between more specific and fine-tuned activities such as reading a book and playing a game on a smart phone, or drinking from a cup, more sophisticated and high-dimensional measurements seem to be needed, along with training on deeper and more expressive neural networks. A one-layer LSTM network performed the best for my collected 6-dimensional data. This says something about the sequentiality being more prominent than possible hierarchical patterns in this kind of data.

Bibliography

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[2] M.S Tremblay, A.G. LeBlanc, M. E. Kho, and T.J. Saunders. Systematic review of sedentary behaviour and health indicators in school-aged children and youth. *Int J Behav Nutr Phys Act*, 8(98), September 2011.

[3] Treuth MS, Baggett C, and et al Pratt C. A longitudinal study of sedentary behavior and overweight in adolescent girls. *Obesity (Silver Spring, Md)*, doi:10.1038/oby.2008.598:1003–1008, 2009. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3739452/.

[4] Licheng Zhang, Xihong Wu, and Dingsheng Luo. Recognizing human activities from raw accelerometer data using deep neural networks. In *14th IEEE International Conference on Machine Learning and Applications, ICMLA 2015, Miami, FL, USA, December 9-11, 2015*, pages 865–870, 2015. doi: 10.1109/ICMLA.2015.48. URL https://doi.org/10.1109/ICMLA.2015.48.

[5] Nils Y. Hammerla, Shane Halloran, and Thomas Ploetz. Deep, convolutional, and recurrent models for human activity recognition using wearables. *ARXIV*, eprint arXiv:1604.08880, 2016. URL https://arxiv.org/abs/1604.08880.

[6] FJ Ordonez and D. Roggen. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16(1), January 2016.

[7] Opportunity activity recognition challenge. http://www.opportunity-project.eu/challe 2011. Accessed: 2017-04-14.

[8] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *CoRR*, abs/1503.04069, 2015. URL http://arxiv.org/abs/1503.04069.

[9] A. Reiss and D. Stricker. Introducing a new benchmarked dataset for activity monitoring, 2012.

[10] Sinziana Mazilu, Michael Hardegger, Zack Zhu, Daniel Roggen, Gerhard Tröster, Meir Plotnik, and Jeffrey M. Hausdorff. Online detection of freezing of gait with smartphones and machine learning techniques. In *PervasiveHealth*, pages 123–130. IEEE, 2012. ISBN 978-1-4673-1483-1. URL http://dblp.uni-trier.de/db/conf/ph/ph2012.htmlMaziluHZRTPH12.

[11] Yago Saez, Alejandro Baldominos, and Pedro Isasi. A comparison study of classifier algorithms for cross-person physical activity recognition. *Sensors*, 17(1), 2017. ISSN 1424-8220. doi: 10.3390/s17010066. URL http://www.mdpi.com/1424-8220/17/1/66.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory, 1995.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9 (8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[14] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, pages 5–6, 2005.

[15] Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins. Learning to forget: Continual prediction with lstm. *Neural Comput.*, 12(10):2451–2471, October 2000. ISSN 0899-7667. doi: 10.1162/089976600300015015. URL http://dx.doi.org/10.1162/089976600300015015.

[16] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer, 2012. ISBN 978-3-642-24796-5.

[17] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020, 9780262018029.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional

[19] Large scale visual recognition challenge 2010. http://image-net.org/challenges/LSVRC/2010/ind 2010. Accessed: 2017-03-03.

[20] David G. Lowe. Object recognition from local scale-invariant features, 1999.

[21] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *In CVPR*, pages 886–893, 2005.

[22] Barret Zoph Quoc Le. Using machine learning to explore neural network architecture. https://research.googleblog.com/2017/05/using-machine-learning-to-explore.html, 2017.

[23] J. Bayer. *Learning Sequence Representations*. PhD thesis, Technischen Universität München, 2015.

[24] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014. URL http://arxiv.org/abs/1404.7828.

[25] Yann LeCun, Geoffrey Hinton, and Yoshua Bengio. Deep learning. *Nature*, 521(7553): 436–444, 5 2015. ISSN 0028-0836. doi: 10.1038/nature14539.

[26] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Netw.*, 2(5):359–366, July 1989. ISSN 0893-6080. doi: 10.1016/0893-6080(89)90020-8. URL http://dx.doi.org/10.1016/0893-6080(89)90020-8.

[27] H. W. Lin, M. Tegmark, and D. Rolnick. Why does deep and cheap learning work so well? *ArXiv e-prints*, aug 2016.

[28] M. Belkin. *Problems of Learning on Manifolds*. PhD thesis, The University of Chicago, 2003.

[29] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. http://www.neuralnetworksanddeeplearning.com.

[30] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166, March 1994. ISSN 1045-9227. doi: 10.1109/72.279181. URL http://dx.doi.org/10.1109/72.279181.

[31] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012. URL http://arxiv.org/abs/1211.5063.

[32] Christopher Olah. Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015.

[33] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/. Software available from tensorflow.org.

[34] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[35] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL http://arxiv.org/abs/1605.02688.

[36] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[37] Opportunity project information. http://www.opportunity-project.eu/project_infor 2011. Accessed: 2017-03-30.

[38] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*, 2010.

[39] Frank Hutter, Holger Hoos, and Kevin Leyton Brown. An efficient approach for assessing hyperparameter importance. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 754–762, 2014. URL http://jmlr.org/proceedings/papers/v32/hutter14.html.

[40] Hrushikesh Mhaskar, Qianli Liao, and Tomaso A. Poggio. Learning real and boolean functions: When is deep better than shallow. *CoRR*, abs/1603.00988, 2016. URL http://arxiv.org/abs/1603.00988.

[41] Tomaso A. Poggio, Hrushikesh Mhaskar, Lorenzo Rosasco, Brando Miranda, and Qianli Liao. Why and when can deep - but not shallow - networks avoid the curse of dimensionality: a review. *CoRR*, abs/1611.00740, 2016. URL http://arxiv.org/abs/1611.00740.

Appendix A

A.1 Raw data

Figure A.1: *Plots of the raw data from categories 0-3. 20 second excerpts.*

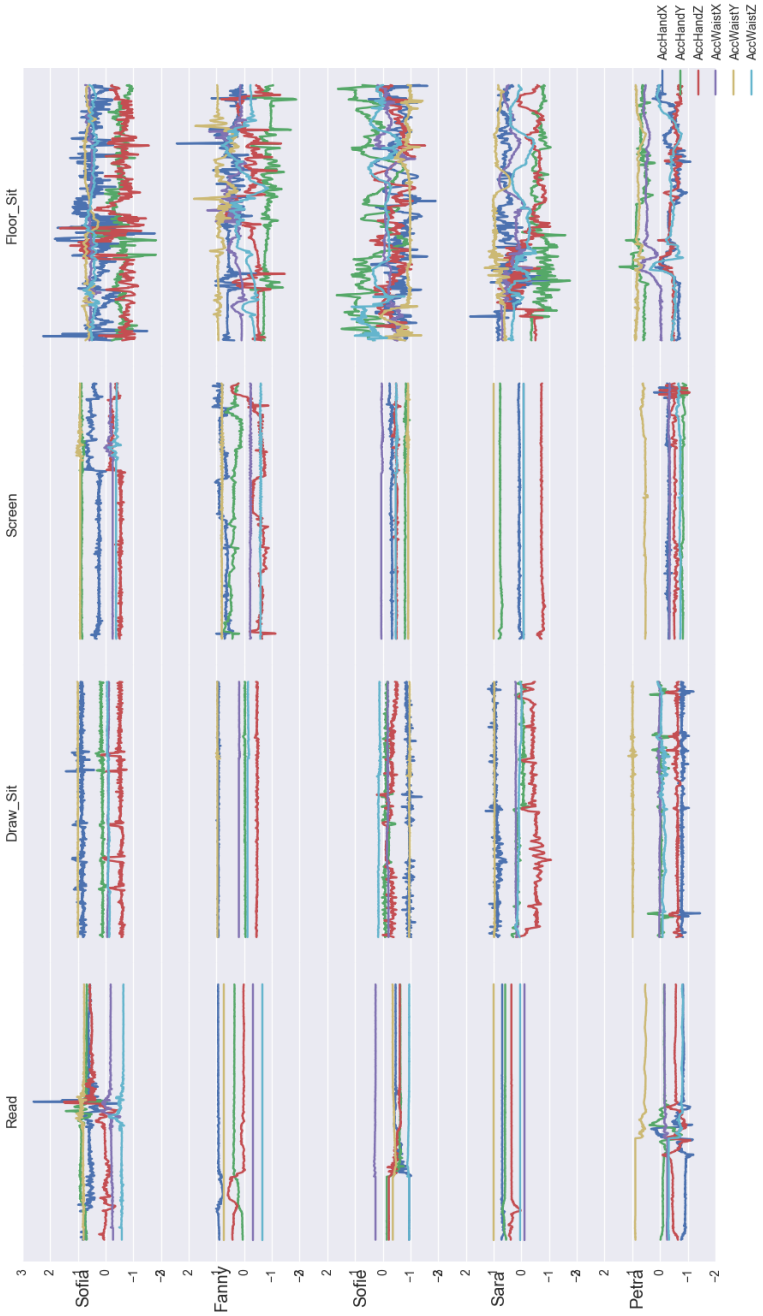
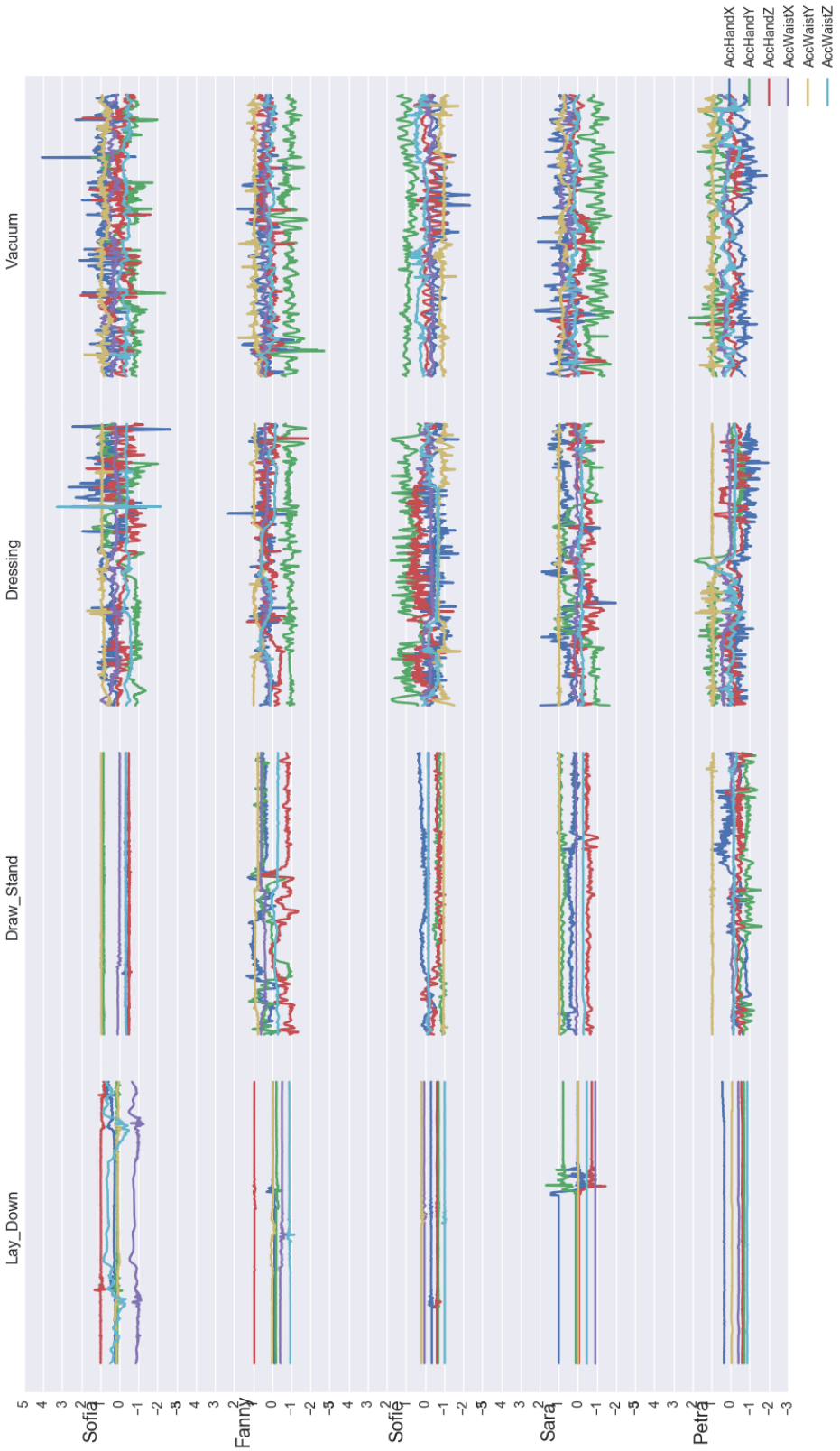


Figure A.2: *Plots of the raw data from categories 4-7. 20 second excerpts.*



A.2 Model specifications

A.2.1 Parameter settings

Table A.1: *Specification on the best performing setups for each architecture. WS is window size, BS is batch size, LR is learning rate, HU is number of hidden units, E is maximum number of epochs, ESP is early stopping patience.*

Parameter	Baseline FC	Baseline CNN	Baseline LSTM	DCLSTM	B-LSTM
WS	30	30	60	30	30
BS	100	100	100	100	100
LR	0.001	0.001	0.001	0.001	0.01
HU	100	100	200	128	128
E	500	500	500	500	500
ESP	100	100	100	100	100

A.3 Number of parameters

Table A.2: *Number of parameters for the different models in A.1.*

Data	Baseline CNN	Baseline LSTM	Baseline FC	DCLSTM	B-LSTM
KI-4	2 244	166 404	28 604	294 532	138 772
KI-5	2 309	166 605	28 705	294 661	138 905
KI-12	2 764	168 012	29 412	295 564	139 836
OPP w/o null	36 484	252 004	349 604	328 772	248 340
OPP	36 549	252 205	349 705	328 901	248 473

A.4 Optimization

A.4.1 The Adam algorithm

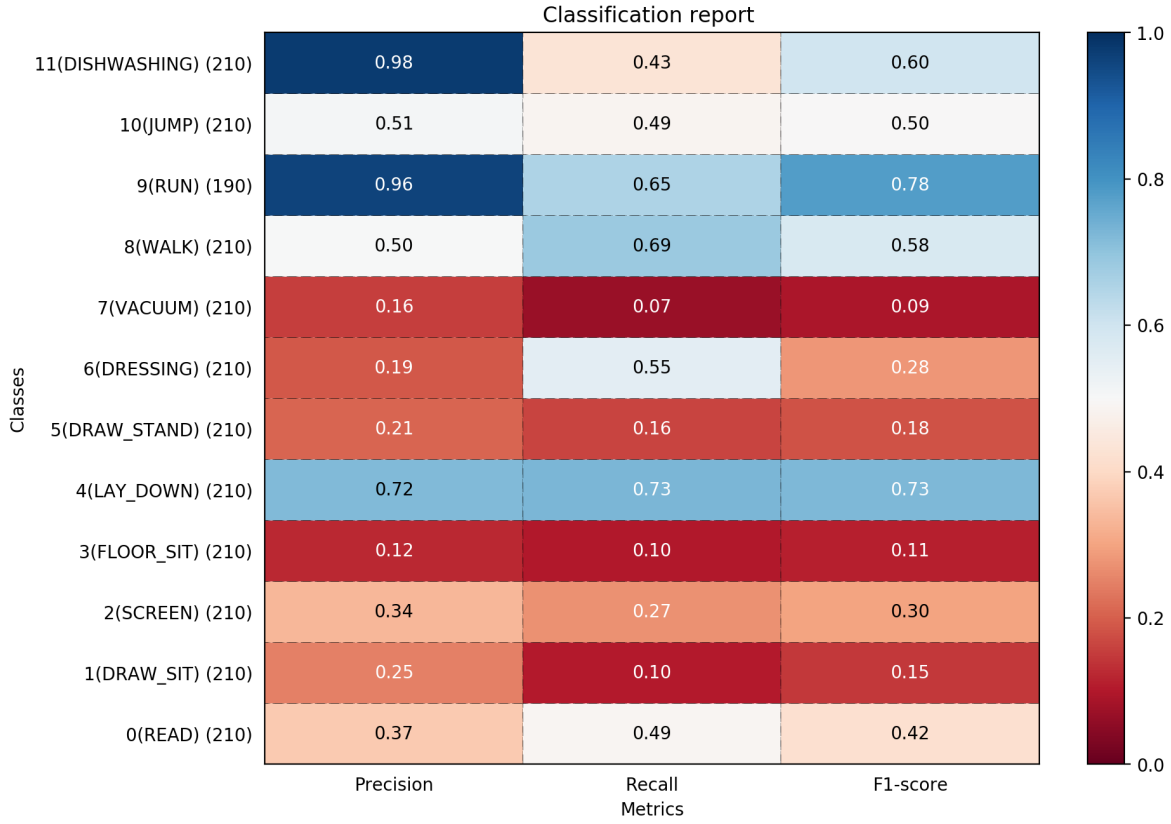
Algorithm 0: Adam [1]

Require: Step size ϵ
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$.
Require: Small constant δ used for numerical stabilization
Require: Initial parameters θ
Initialize 1st and 2nd moment variables, $\mathbf{s} = \mathbf{0}, \quad \mathbf{r} = \mathbf{0}$
Initialize time step $t = 0$

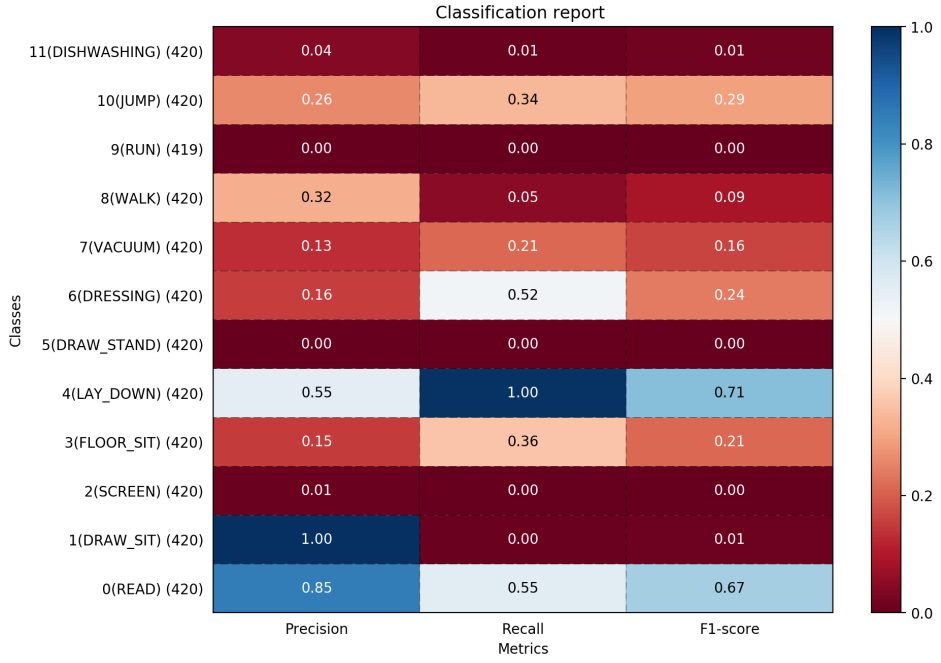
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.
Compute gradient $\mathbf{g} = \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} L(\mathbf{x}^{(i)}, y^{(i)}, \theta)$
$t' = t + 1$
Update biased first moment estimate: $\mathbf{s}' = \rho_1 \mathbf{s} + (1 - \rho_1)\mathbf{g}$
Update biased second moment estimate: $\mathbf{r}' = \rho_2 \mathbf{r} + (1 - \rho_2)\mathbf{g} \odot \mathbf{g}$
Correct bias in first moment: $\hat{\mathbf{s}} = \frac{\mathbf{s}}{1-\rho_1^t}$
Correct bias in second moment: $\hat{\mathbf{r}} = \frac{\mathbf{r}}{1-\rho_2^t}$
Compute update: $\Delta\theta = -\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}} + \delta}$
Apply update: $\theta' = \theta + \Delta\theta$

end while

A.5 Results



(a) Baseline LSTM, adult population



(b) DeepConvLSTM, adult population

Figure A.3: *The precision, recall and F1-scores on the 12-class task of KTH-KI-AA for the baseline LSTM and the DeepConvLSTM. The numbers to the right of the category names in 4.1a are the number of samples per class in the test set. The baseline LSTM has a higher overall score, but the point is to note how uneven the DeepConvLSTM is in its performance across categories.*