Imperial College London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF EARTH SCIENCE AND ENGINEERING

MSC APPLIED COMPUTATIONAL SCIENCE AND ENGINEERING
INDEPENDENT RESEARCH PROJECT

Generation of multi-featured time-series sensory data

Author:
Hameed Khandahari
github.com/hk-97

Email:
hk2715@ic.ac.uk

CID:
01069638

External Supervisor:
Dr H. Haddadi
Brave Software

Internal Supervisor:
Dr G. Gorman

Submitted in partial fulfillment of the requirements for the MSc degree in Master of Science of Imperial College London

September 2019

Abstract

A Python package was developed to produce a generative adversarial model capable of generating realistic time-series sensory data. The model was trained on the `MotionSense` dataset. This dataset consists of sensory data from twelve different features, captured when the test subjects were performing one of six activities. The features used for this project were limited to only accelerometer data, along three orthogonal axes. A generative adversarial network, consisting of deep convolutional discriminator and generator networks was used for the data generation. When visualised, the data generated appears to resemble that from the dataset, and when passed through the classification model provided by the authors of `MotionSense`, the best class (data associated with jogging) obtained an accuracy of 95 %, while other classes scored poorly.

Acknowledgments

I would like to thank all members of the Department of Earth Science and Engineering that have played a role in running this masters programme for its excellent teaching and organisation. Specifically, I thank Dr Gorman for establishing the programme and Dr Paluszny, Dr Neethling and Dr Collins, whose courses I have particularly enjoyed. Equally, I would like to thank my peers for their friendship and humour throughout the last twelve months.

Specifically for this project, I would like to thank my supervisor, Dr Haddadi, for proposing and motivating the project, and Mr Malekzadeh, for providing support and guidance.

Contents

Chapter 1

Introduction

1.1 Motivation

This work was proposed by the supervisor for this project, Dr. Haddadi. Dr Haddadi is a a Senior Lecturer in the Dyson School of Design Engineering at Imperial College and is currently working with the research division at *Brave*[1]. Founded in 2015, *Brave* are a new, privacy-focused web browser which is seeking to introduce a better web browsing ecosystem by using, amongst other technologies, developments in deep learning. The new ecosystem proposed by *Brave* includes features such as the automatic detection and blocking of advertisements, by using graph-based machine learning techniques [13]. It also addresses the issue of web trackers, which are a major privacy concern, by providing automatic tracker blocking and fingerprinting prevention [1].

Of particular interest to this project, however, is their proposed advertising model. In the current advertising ecosystem, revenue from advertising is collected by the websites on which they are hosted [10]. In the *Brave* ecosystem, advertisements are blocked by default, but users have the option of opting to allow advertisements, in which case they are rewarded for their 'attention' with BAT (Basic Attention Token) tokens, an *Ethereum* based cryptocurrency[9], which can then be converted to conventional currency.

This can however, be open to exploitation if not implemented with caution. Even in the current ecosystem, issues such as clickfarming [6][8] have become prevalent; criminals can host advertisements on websites, and then use bots to interact with the advertisements to generate revenue for themselves. Activity such as this can be detected, by systems such Google's `reCAPTCHA v3`[3], which is run in the background during web activity and uses the data it is able to gather about a visitor to return an estimate of how likely they are to be authentic. In the proposed ecosystem, the number of users who may attempt to earn revenue in this manner will likely increase, since it would no longer be required to host advertisements on websites; instead all that is required is to fool the detection algorithms,

Therefore, steps must be taken to minimise the potential impact by preparing for more sophisticated fraudulent activity, which would be able to bypass current detection techniques.

1.2 Defining this project

The aim of this project is not to directly address the issue detecting unauthentic activity itself, but instead to seek to investigate whether it is possible to generate highly realistic time-series sensory data (such as accelerometer data), which would be able to fool activity recognition models.

The tools, with which this is attempted are based on recent developments in generative models in the field of deep learning. In particular, generative adversarial networks, which have been very successful in domains such as imaging, are applied to time-series data to see whether realistic data can be generated.

1.3 Further applications

The underlying architecture that is developed should be, in principle, applicable to a wide range of data that exhibits one-dimensional sequentiality, as time-series sensory data does. A particularly interesting use case, where much work has been done [12] in the context of medical data, is for privacy protection matters. It would for example be possible, to generate synthetic medical data, which would otherwise be difficult to obtain due to privacy laws, that can be useful for training further models.

Chapter 2

Background of models

In this Chapter, the reasons for deciding to use generative adversarial networks as the generative model of choice is described. They are then formally introduced through considering the mathematics that underpins them.

2.1 Selecting generative models

Developments in machine learning, specifically generative models in deep learning, over the last few years have allowed data generation in fields such as music [17] and text [20] production through the use of recurrent neural networks; they have also been able to generate ultra realistic images, which are almost indistinguishable from the data with which they are trained, using generative adversarial networks(GANs).

However, much of the work completed concerning GANs, has focused on image generation. These models are based on deep convolutional neural networks which have now been well established [18] [22]. There has however, been limited work done on the generation of one-dimensional data [17] [12] using GANs.

If the problem of data generation in n-dimensions is considered, it may appear that the one-dimensional case should be trivial if the two-dimensional (for images)[19], and to some extent the three dimensional [21] (two spatial and one temporal dimension for videos) cases have been solved successfully. However, due to the nature of one-dimensional data, the tools which have been instrumental in success of image generation GANs are no longer as powerful.

The Deep Convolutional GAN (DC-GAN) [19], upon which many successful image based GANs are built upon, illustrates this well. DC-GANs contain a series of two-dimensional convolution layers, which allows the neural network to contextualise and localise information using information along two dimensions. This is not possible for the one-dimensional case since convolutional kernels would, by definition, be restricted to one dimension. Therefore, it is worth investigating to find a neural network architecture that can accommodate one-dimensional data well.

2.2 Generative Adversarial Networks (GANs)

A generative adversarial network is comprised of two competing neural networks: a generator, which produces an output; and a discriminator, which assesses the quality of this output [11]. Through the combined training of these two networks, it is hoped that the generator is trained to such a degree, that the output that it produces appears indistinguishable from the data in the training set.

2.2.1 The Discriminator

The discriminator is a neural network that is trained to distinguish between real data (taken from the training set) and fake data (taken from the output of the generator). This neural network is designed such that it has a single output, a value between `0` and `1`, indicating the probability that the input is from the real data. The architecture of the discriminator is very similar to that of a classifier, such as those used to classify the 10 classes of digits in the `MNIST` dataset. However, the architecture used here allows for only two *classes*, real and fake, hence it does not distinguish between different classes of real data, but seeks only to determine how likely they are to be real.

2.2.2 The Generator

The generator is a neural network which is trained to produce data similar to that in the training set. It takes a randomly initialised vector, the '*latent space vector*' as its input, and transforms it into a data sample, through a series of upsampling processes. If the underlying architecture of the generative neural network has been designed well, it is hoped that the training process will result in a generator that is able to convert any latent space vector into a data sample that is indistinguishable from the training set. The architectural design choices of both the discriminator and generator neural networks are described in Section 3.7.

2.2.3 Training Process

The complete generative adversarial network model consists of these neural networks being trained in tandem. The discriminator is a binary classifier, trained with correctly labelled samples from the training set and the generators output. The generator, itself, is trained *using* the discriminator. It is trained to produce outputs such that when passed through the discriminator, the discriminator incorrectly identifies the generated samples as being from the training set.

Hence, the discriminator is trained to reject generated samples, and the generator is trained to produce samples whose distributions mimic that of the training set such that the discriminator is fooled.

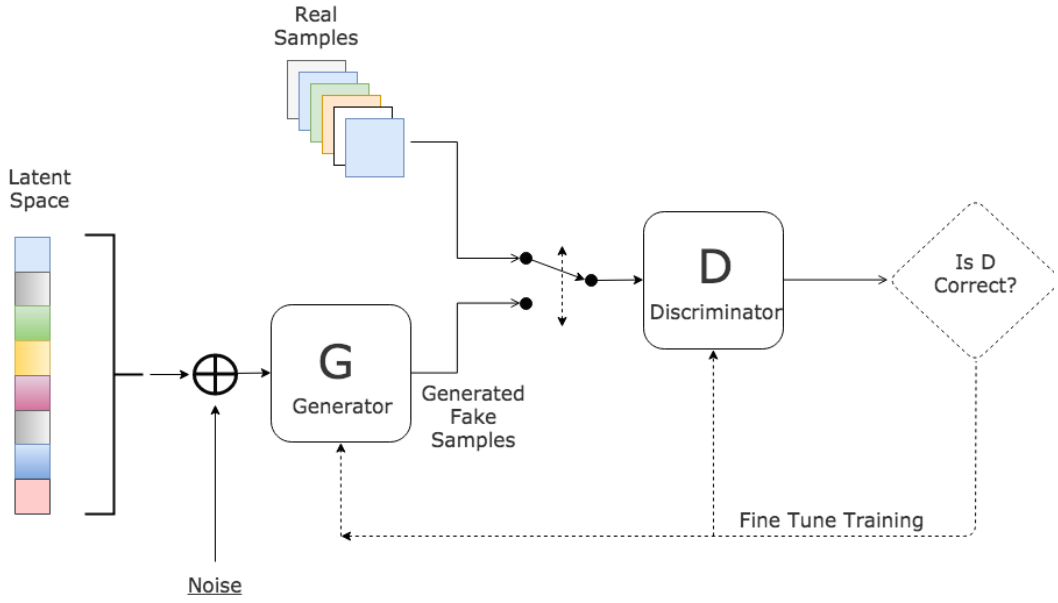

Figure 2.1: Schematic of the fundamental architecture of a GAN[2]

An appropriate loss function used to calculate the quality of the discriminator is the binary cross entropy function, since it is performing a binary classification task. By extension, since the quality of the generator is calculated by the performance of its output on the discriminator, the loss function associated with the quality of the generator is also a binary cross entropy function. This is given by

$$L\left(\hat{y}, y\right) = -\Big[y\, log\hat{y} + (1-y)\, log(1-\hat{y})\Big], \tag{2.1}$$

where y and $\hat{y}$ are the known and predicted labels of data in the context of a classification task.

Training the Discriminator

In order to minimise the binary cross entropy loss for the discriminator, it must perform well on correctly classifying both real and fake data as such. For real data, $\boldsymbol{x}$, the assigned label is given as 1 and the predicted label from the discriminator, $\boldsymbol{D}$, is given as $\boldsymbol{D}(\boldsymbol{x})$. For fake data, the assigned label is 0 and the predicted label is $\boldsymbol{D}(\boldsymbol{G}(\boldsymbol{z}))$, where $\boldsymbol{G}$ is the generator and $\boldsymbol{z}$ is the latent space vector. Applying these conditions to the binary cross entropy function, we are left with the following two optimisation criteria,

$$\min_{\boldsymbol{D}} \; -log(\boldsymbol{D}(\boldsymbol{x})), \tag{2.2}$$

$$\min_{\boldsymbol{D}} \; -log(1-\boldsymbol{D}(\boldsymbol{G}(\boldsymbol{z}))). \tag{2.3}$$

Training the Generator

As explained, the loss function used for the generator is also a binary cross entropy function, as the objective is to fool the discriminator, which is a binary classifier. The samples from the generator are presented as real, i.e. with a $y = 1$. The score generated by the discriminator is $\boldsymbol{D}(\boldsymbol{G}(\boldsymbol{z}))$. The loss associated with this is to be minimised,

$$\min_{\boldsymbol{G}} \; -log(\boldsymbol{D}(\boldsymbol{G}(\boldsymbol{z}))). \tag{2.4}$$

Reformulating 2.4 to make it compatible with 2.3, and putting these sets of optimisation metrics together into a compact form, leads to the form introduced by Goodfellow in the landmark 2014 GAN paper, [11]

$$\min_{\boldsymbol{G}} \max_{\boldsymbol{D}} V(\boldsymbol{G}, \boldsymbol{D}) = log\boldsymbol{D}(\boldsymbol{x})) + log(1 - \boldsymbol{D}(\boldsymbol{G}(\boldsymbol{z}))), \tag{2.5}$$

where $\boldsymbol{G}$, $\boldsymbol{D}$ are the generator and discriminator functions respectively, $\boldsymbol{x}$ is a sample taken from the training set and $\boldsymbol{z}$ is a random latent vector used by $\boldsymbol{G}$ to map into the space of $\boldsymbol{x}$.

Thus the statement that the two neural networks are competing with each other is mathematically formalised.

Chapter 3

Implementation

In this Chapter, the way in which the programme was developed is described and details of the frameworks used are explained. The design choices in the various networks are justified and a description of how the models evolved during the project are given.

3.1 Software Development Life Cycle

A Software Development Life Cycle (SDLC) consists of a series of procedures carried out during a software development project to ensure that the aims and the scope of the project are well understood and the best ways in which they can be achieved are determined [5]. Several SDLCs were considered for this project and it was decided that the 'Spiral' method is best suited for this project. Here, the features of the most popular frameworks are analysed briefly to explain the choice made.

3.1.1 Comparison of widely used models

Three of the most widely used models are the 'Waterfall method', the 'Spiral method' and the 'Agile method' [5].

The 'Waterfall method', which consists of rigid, sequential phases, whereby one phase must be realised to completion before the succeeding phase can begin was deemed too rigid for this project. This model is best suited to projects with a single main goal and leaves little room for adaptation of the initial plan. The Agile method, does offer the flexibility which the Waterfall method lacks, however it is more suited to commercial software development projects which is performed by teams for a commercial client. Therefore, it was the 'Spiral method', which is best suited for a research-oriented project such as this, since it offers the flexibility of the Agile method, whilst allowing progressively more complicated 'products' to be developed within the timeframe of the project.

3.2 Package Design

An object-oriented design was selected for the project since this lends itself well to the modularity that object-oriented design allows. This design allows the production of a package which is easily maintainable and to which further functionality can later be added.

The data itself has been obtained from the MotionSense GitHub repository, and is provided within the repository of this project in the `MotionSense/` folder. The `gan.py` file includes a python class containing for the generative adversarial network. It contains methods to build the discriminator, the generator and then to combine these to build the GAN. Furthermore, it includes functionality to use the generative model to produce a sample of data. The `trainer.py` file includes a python class containing methods for training instances of the GAN class. It also has functionality for saving the progress of the training process and information regarding the loss at each epoch. The `DataLoader.py` file includes a python class containing methods for loading data from the `.csv` files in the `MotionSense\` folder and restructuring them such that they can be used to train the models. These methods have been adapted from code in the original MotionSense repository. The `DataPreparation.py` file includes a python class contains routines used for the extraction and restructuring of data. It contains methods which use the naming system used by the authors of the `MotionSense` dataset to parse through the data files within the dataset and to collate arrays containing only the requested data in a form which can be used for training the neural networks. These methods have been adapted from existing code on the `MotionSense` repository. The `DataHandler.py` file includes a python class containing methods for feature scaling (normalisation and standardisation and corresponding reverse transformations). The `utils.py` file includes helper functions to create directories for storing data and images, obtaining random batches of data and plotting results.

3.3 Code metadata

This project was completed using Python v3.7, a high-level programming language, with a number of popular libraries available for machine learning. The library of choice for this project was `Keras`. Although the machine learning module in this master's programme was taught primarily with `PyTorch`, the underlying philosophies in the construction of the neural networks remains unchanged. The decision to complete the project using a a new, unfamiliar library was taken because the original work done on the `MotionSense` project, which can be found on the `MotionSense` repository was developed using `Keras`. Hence, I was able to integrate the existing code into this project. This is discussed further in Section 3.5.

The code was developed primarily on Google Colab. This is a cloud-based iPython Notebook `.ipynb` platform, which allows access to a graphical processing unit (GPU) which gave a significant increase in time efficiency. This is due to the capability of the GPU to parallelise matrix multiplication calculations, which is what the training of neural networks fundamentally depends upon. Furthermore, Colab

offers version control by allowing access to revision history. The code is available at `https://github.com/msc-acse/acse-9-independent-research-project-hk-97`.

3.4 Training data

3.4.1 Sources of data

The sensory data used for training the models is taken from the `MotionSense` dataset [15]. This dataset, contains time-series sensory data recorded using an `iPhone 6S` by utilising the `SensingKit` mobile sensing framework developed at Imperial College. The dataset itself was gathered by Mr. M. Malekzadeh, an advisor for this project. It consists of measurements from 12 sensors, across 15 experiments, by 27 candidates performing 6 activities.

3.4.2 Exploratory Data Analysis

The data used for this project was first explored to cement understanding of its features, usability and limitations. An example of the data, taken from an event where jogging is taken place is shown in Figure 3.1. The models developed in this project have so far only been trained with accelerometer data, the temporal measurements of which along 3 orthogonal directions are given in Figue 3.1. The project was deliberately limited to accelerometer data, since it was noted that using a classifier with only accelerometer data is sufficient to correctly identify the activity being peformed in 98 % of cases, when tested on an unseen test set (Section 3.5). Furthermore, limiting the number of activities reduces the training time, this time was spent trying to improve the architecture of the GAN itself.

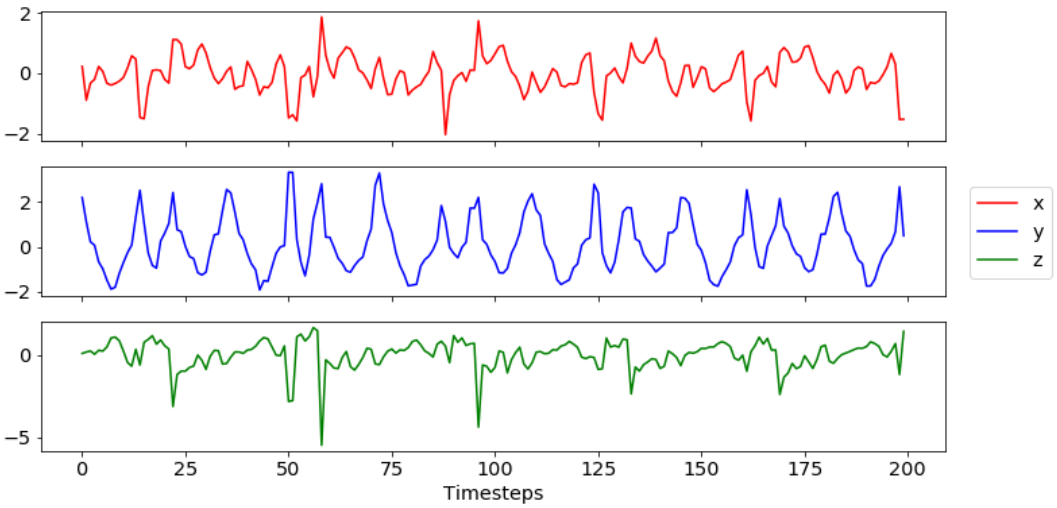

Figure 3.1: Caption

Observing the triaxial accelerometer data, several features can be noted. First of all, there is some degree of regularity in the data for each of the axes. This is to be

expected since jogging is an activity which should produce a signature trace at regular intervals as steps are taken. The primary mode of this 'stepping' appears to be captured best by the `z` axis, as a sudden jump in the accelerometer data is observed approximately once every 30 timesteps (0.6 seconds) which seems reasonable for the time between steps, which are just impacts on the ground, whilst jogging. The `y` axis appears to be capturing another more regular and frequent mode of oscillation, which could have any number of explanations concerning the internal body movements of the subject.

3.4.3 Data preparation

Extraction

The dataset has been obtained from the `MotionSense` repository and has been provided in the repository of this project in the `\MotionSense` folder. This folder is further divided into 15 different folders, one for each experiment of the experiments described above. The folders contain 27 `.csv` files, each containing twelve streams of sensory data captured by individual participants in the data collection trials. Existing code, taken from the `MotionSense` repository, was used to parse these files to generate training and test sets for each activity. Furthermore, this code was adapted to add further functionality and then integrated into the object-oriented design of this work.

The training sets are prepared by collating the data gathered across all experiments into a series of arrays, with each array containing data taken from just one of the six (`dws`, `ups`, `std`, `sit`, `wlk`, `jog`) activities. Each of these arrays are of shape (n,1), where n is the number of timesteps. To prepare the training set, these arrays are sliced into smaller sections, with each section acting as one element of the training set. This is done using a sliding window approach.

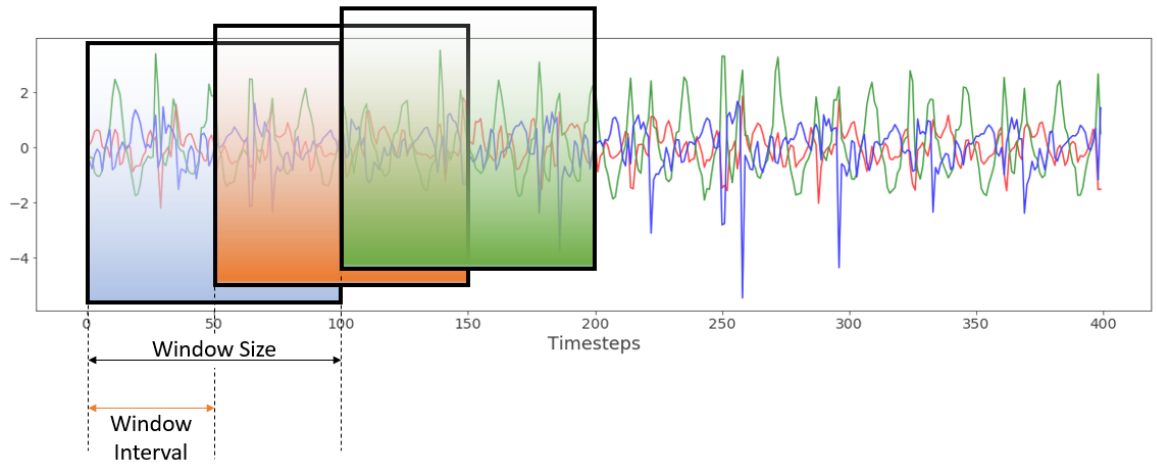

Figure 3.2: The sliding window approach used to segment the data into smaller chunks. The colored rectangles represent the sliding windows. Only three windows have been shown in the Figure; however this is just to avoid clutter. The window size and interval shown in the Figure are for illustrative purposes only.

Whilst, twelve streams of sensory data were available in the dataset, only the accelerometer data, along three axes, was considered for the quantitative parts of this project, since it was realised that classification can be achieved to a very high level using only this data.

Feature scaling

Two different feature scaling techniques were used in this project; these are min-max normalisation and z-score standardisation. It should be noted that the nomenclature concerning this topic varies amongst the literature.

In this report min-max normalisation is defined to be,

$$\bar{\boldsymbol{x}} = 2\,\frac{\boldsymbol{x} - x_{min}}{x_{max} - x_{min}} - 1, \tag{3.1}$$

where $\bar{\boldsymbol{x}}$ is the data after the scaling, $\boldsymbol{x}$ is the data before the scaling and x_{min} and x_{max} are the minimum and maximum values in the data, prior to scaling, respectively.

Secondly, z-score normalisation is defined to be,

$$\bar{\boldsymbol{x}} = \frac{\boldsymbol{x} - \mu}{\sigma}, \tag{3.2}$$

where σ and μ are the standard deviation and mean of the data, prior to scaling, respectively.

The choice of feature scaling used depends on the exact use case. When preparing the classifier, initially standardisation was used, following the original Malekzadeh implementation [16]. However, since normalisation also resulted in high classification scores, this was preferred as it can better be integrated with the generative models. The generator network contains a `tanh` activation function, this limiting the range of the data between -1 and 1, the same range as min-max normalised data.

3.5 Classification

Although building an activity recognition model does not fall under the remit of this project's objectives, utilising such a model will help to quantify the performance of the generative models.

The model developed by the authors of the `MotionSense` dataset, for a work on Data Sensor Anonymisation [15] was used for this, and has been integrated into this project. The model takes as its input, an array containing time series sensory data from the training set, and returns a score indicating how likely each activity was to have produced such data. The model performs well on the original `MotionSense` data, obtaining accuracy scores of 98.3 % accuracy on the test set.

The intention is to run the data which has been generated from the generative models, through the pre-trained activity recognition model, to see whether there is an agreement between the label of the training data which has produced the new data, and the prediction by the classification model.

3.6 Hyperparemeter selection

The stability of these models are dependent on the choice of hyperparameters. Hyperparameters remain invariant during the training process and must be tuned to improve the performance and stability of the neural networks.

3.6.1 Choice of optimiser

The most crucial hyperparameter proved to be the choice of the optimiser. Optimisers are crucial to the training of a neural network as they control how the weights are adjusted during the training process. The choice of optimiser, and parameters thereof, proved to be a useful tool in tuning the relative 'strength' of each competing neural networks to ensure that no neural network was over-trained. The `Trainer` module has been designed such that the optimisers for both the discriminator (or critic) and generator are stored as attributes of instances of the `Trainer` class. This design choice was made to provide a way of easily adjusting this hyperparameter.

For the standard GAN implementation, initially the `Adam` optimiser was used with the learning rate of 0.001, as suggested in the Adam paper [14], where it was used in the context of training a model on the MNIST dataset. Line searches were made around this value to find more suitable parameters for this specific problem.

However, using `Adam` as the optimiser for both the discriminator and generator networks proved to be very unstable. Despite performing a grid search over the learning rates for each optimiser, no sufficiently stable configuration was found. However, changing the optimiser for the discriminator to a stochastic gradient descent (`SGD`), as suggested by a Deep Learning engineer at Facebook [4] allowed a stable configuration to be found after tuning the learning rate parameter for the `SGD`, whilst keeping the learning rate for the `Adam` generator optimiser as 0.002 .

3.6.2 Sequence Length

In this context, the sequence length refers to the length of the time dimension in both the training and generated data. Due to the nature of how the neural network architectures have been constructed, these two lengths are required to be the same.

Thus, changing how the training data is prepared, using the `DataLoader` class, to produce a training set with a different sequence length, will result in a change in the sequence lengths of the generated samples. However, since the discriminator network begins with a convolutional layer, the actual architecture of the discriminator network itself will need to be tuned, which could result in unpredictable changes. This could be resolved by replacing the the first layer of the network with a fully connected, dense layer, such that the structure of the network will be invariant for all input sizes. However, in this investigation, the size of the sequence length was fixed at 200 timesteps, which translates to 4 seconds. This choice was made since it was reasoned that 4 seconds of sensory data, should contain enough information to determine the activity being performed.

Furthermore, the architecture of the neural networks themselves can also be considered to be a collection of hyperparameters; since, while the weights and biases of the networks are updated through the training process, the underlying architecture does not change. This is explored further in the next section.

3.7 GAN Architecture

Several implementations of the standard GAN were considered. The model provided in the repository, and the one from which results in this document have been produced, was the most stable implementation found for this data. It consists of deep convolutional discriminator and generator networks. The architecture for these, in particular the discriminator, was highly based on the activity recognition network since it was known to be effective for this data structure, considering the high classification score obtained, even when using just three features (accelerometer data).

The discriminator consists of one-dimensional convolutions along the data for each feature, i.e. each axis of accelerometer data followed by a dense layer which combines the data from different features. It is this dense layer which allows the relationship between the different features to be captured. The generator, also makes use of convolutional layers, however in this case it is to upsample the one dimensional latent vector into data representing each of the features. The GAN architectures developed without convolutional layers, i.e. fully connected neural networks, were found to be more prone to mode collapse and hence these models were discarded.

3.8 Training process

The instances of the GAN model classes contains the the discriminator and generator networks as attributes. These instances are passes through the Trainer class which contains methods for training the combined network. Whilst nomenclature in machine learning literature, 'epoch' usually refers to a training cycle which has seen all of the training data, here Goodfellow's original implementation, where only one batch is used per 'epoch' is used [11].

3.8.1 Instability of GAN training

GANs are notoriously difficult to train due to their inherent instability. If the discriminator is too strong, then all of the generated samples will be correctly identified as being fake. Alternatively, if the generator appears to be too strong in relation to a weak discriminator then the gradients in the generator will be too low for any optimisation to take place.

There has been some work done on improving the stability of GAN [4]. Some of these techniques were implemented. These techniques include label smoothing, whereby the assigned label that is passed through the discriminator is stochastically adjusted with some noise. This was implemented but little if any improvement was noted.

3.9 Integration testing

Integration tests were completed after the functions from the `MotionSense` repository were incorporated into this project and its object oriented design. This was done by retraining the classifier in the `MotionSense` repository, using the data preparation techniques introduced in the ecosystem used in this project, and ensuring that the results were as expected.

Further unit testing included ensuring the implementation of the feature scaling methods in the DataPreparation module were correct. This was done by performing a forward, then backward scaling, and ensuring that the residual between the original data, and the data that had undergone the scaling, was zero.

Other forms of testing were performed on an ad-hoc basis, this included testing the inputs and outputs of neural networks to ensure that their dimensions were correct.

Chapter 4

Results

In this section, the results obtained from a trained GAN model are presented. These quality of the results, especially the variance of the quality of output produced for each activity, is used to gain insight about the strengths and weaknesses of the model that has been developed. Furthermore, the intermediary results: the information obtained during the training process is used to validate that the behaviour of the generator, the discriminator and the the combined network is as expected, based on the theoretical framework developed in Chapter 2.

4.1 Measure of quality

Defining the quality of a GANs output is difficult, since the loss function seeks only to fool the discriminator, not directly to produce a high quality output. More recent GAN developments, such as the Wasserstein (W-GAN) [7] attempt to provide a better measure by using the *Earth-Mover distance* as the loss function, however these have not been successfully implemented in this project.

Generator models have been trained using only training data of single classes. These can be used to produce more data, which mimics that class specifically. The way in which the quality of the GANs has been quantitatively assessed is by passing the generated samples from these models back into the classifier to see whether the classifier returns the expected label.

This achieved mixed success. The generator trained using only jogging data, achieved a score of 95 %, however the other classes achieved below 50 %. This is likely due to the fact that the apparent jaggedness in the results may be fooling the convolutional layers, which would suggest that the architechture of the generator must be revisited.

4.2 Monitoring the Results

Using the `Trainer` module, the losses associated with the discriminator and generator are recorded at a regular intervals specified by the user (every 10 epochs by default). The results of this can be seen in Figure 4.1. We can make several conclusions from this.

Firstly, the initial loss value of the generator is significantly higher than that of the discriminator. This is to be expected, as the discriminator is known to be strong, since it is based on a strong classifier. Therefore, it appears that that after a relatively low number of epochs, the discriminator is able to perform well in correctly distinguishing between the real and generated samples. The generator gradually improves, reflected in the decrease of its loss value. This indicates that the discriminator is starting to struggle to accurately distinguish between the two types of data.

The loss value for both the discriminator and the generator converges to $log2$ after sufficiently many epochs, this is what is expected for an ideal discriminator according to the theory laid out by Goodfellow [11]. Furthermore, the accuracy of the discriminator can also be investigated. This also agrees with the theoretical prediction that the accuracy will converge to 0.5, which implies that the discriminator cannot distinguish between real and fake data.

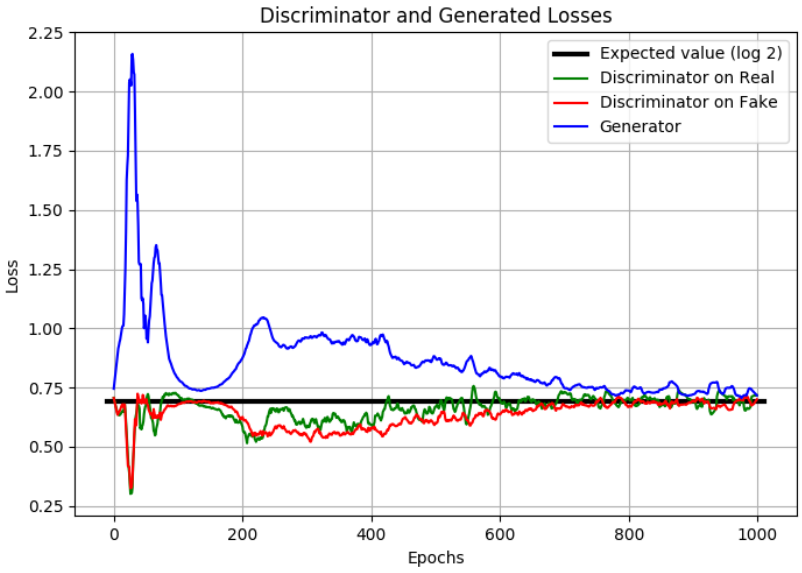

Figure 4.1: Losses per epoch. Smoothed using the scipy savgol filter

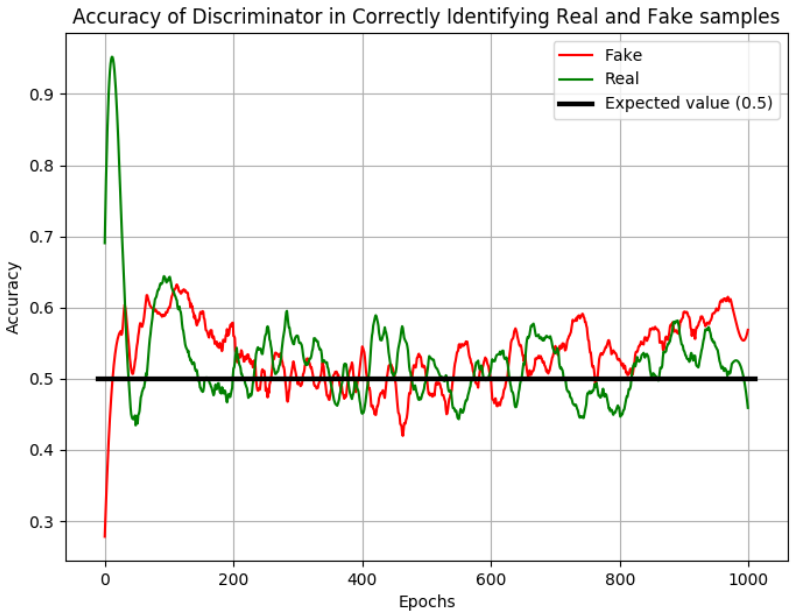

Figure 4.2: Accuracy of the discriminator per epoch. Smoothed using the scipy savgol filter

4.3 Visual Checking

As previously mentioned, there is no single robust measure for assessing the output of a GAN. When using GANs to generate images of real-world objects, a visual check is often useful, since we are very quickly able to recognise a generated image as realistic or not realistic. This task however, is made more difficult for abstract data such as that generated in this project. However, it is still worth an observation. Figure 4.3 contains generated and real samples for the generation of data corresponding to three activities: standing (`std`), jogging(`jog`) and walking (`wlk`).

We can see the varying degrees of perceived quality from the selection of samples in Figure 4.3. The data generated for jogging appears to closely resemble that from the training set. This suggests that the architecture of the GAN is well suited for data such as this, which is relatively regular and periodic.

The data generated for walking however, does not seem to be of such a high quality. This can be explained due to the fact that there is much more variation in walking behaviour, compared to jogging. This claim is made empirically from observing samples of data from the training set labelled as such.

The data generated for sitting is also clearly identifiable as similar to that from the training set, however there are artefacts such as a consistent rise at the boundaries, which are likely to have been cause by the convolutional or max-pooling layers.

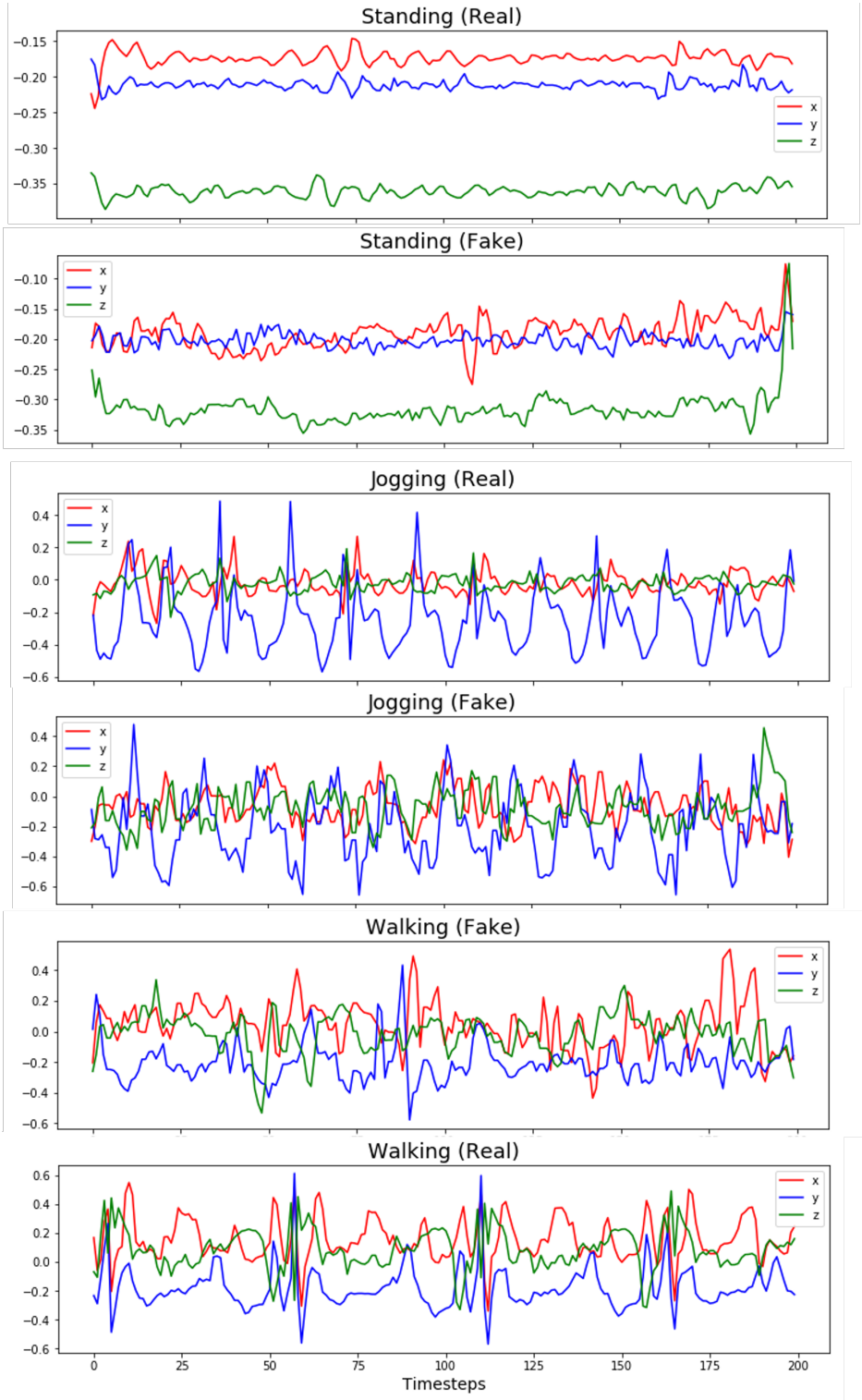

Figure 4.3: Selection of results from the generative model, alongside the corresponding real samples of data

Chapter 5

Discussion and Conclusions

5.1 Difficulties

Generative adversarial networks, in the raw form first introduced by Goodfellow, are known to be very hard to train. This was certainly found to be the case. The deep convolutional generative adversarial network developed appears to produce data that, when plotted, appears visually similar to that from the training set. However, it appears to perform better with some classes of data than others, suggesting the architecture must be further improved

5.2 Future work

Whilst the most successful model developed during this project was a variety of the deep convolutional GAN, it would be interesting to explore other architectures for comparison. Of particular interest is an LSTM GAN implementation, whose structure lends itself well to time series data.

Another interesting field of research could be to explore ways in which the length of sequences generated can be extended arbitrarily. This could potentially be done by training another neural network to join existing sequences together, or by constructing the GAN such that

Adding Conditional GAN. At present, the only way of ensuring that the output of the generative model is of a particular type, is to only use that data to train the model, developing a conditional GAN will eliminate the need for this and streamline the data generation process.

To provide further quantitative analyses of the performance of the GANs, a similar treatment to that prescribed in Section 4.4 could be used with a slight change. Instead of training on the training set, and evaluating a classifier score on the generated samples, it would be interesting to see the results if the classifier model is trained on the generated samples, and the then tested on the training set. This approach is inspired by a work on

Bibliography

[1] Brave Research — Brave Browser. URL: https://brave.com/research/. pages 1

[2] Generative Adversarial Networks . URL: https://www.kdnuggets.com/2017/01/generative-adversarial-networks-hot-topic-machine-learning.html. pages 5

[3] reCAPTCHA v3 — reCAPTCHA — Google Developers. URL: https://developers.google.com/recaptcha/docs/v3. pages 1

[4] soumith/ganhacks: starter from "How to Train a GAN?" at NIPS2016. URL: https://github.com/soumith/ganhacks#authors. pages 12, 13

[5] Adel Alshamrani and Abdullah Bahattab. A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model. Technical report. URL: www.IJCSI.org. pages 7

[6] D. Antoniou, M. Paschou, E. Sakkopoulos, E. Sourla, G. Tzimas, A. Tsakalidis, and E. Viennas. Exposing click-fraud using a burst detection algorithm. In *2011 IEEE Symposium on Computers and Communications (ISCC)*, pages 1111–1116. IEEE, 6 2011. URL: http://ieeexplore.ieee.org/document/5983854/, doi:10.1109/ISCC.2011.5983854. pages 1

[7] Martin Arjovsky, Soumith Chintala, and Lon Bottou. Wasserstein GAN. Technical report. URL: https://arxiv.org/pdf/1701.07875.pdf. pages 15

[8] Anup Badhe. Click Fraud Detection In Mobile Ads Served In Programmatic Exchanges. *INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH*, 5(04), 2016. URL: www.ijstr.org. pages 1

[9] Blockchain Based and Digital Advertising. Basic Attention Token (BAT). Technical report, 2018. URL: https://basicattentiontoken.org/wp-content/uploads/2017/05/BasicAttentionTokenWhitePaper-4.pdf. pages 1

[10] Geumhwan Cho, Junsung Cho, Youngbae Song, and Hyoungshick Kim. An empirical study of click fraud in mobile advertising networks. In *Proceedings - 10th International Conference on Availability, Reliability and Security, ARES 2015*, 2015. doi:10.1109/ARES.2015.62. pages 1

[11] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. Technical report. URL: http://www.github.com/goodfeli/adversarial. pages 4, 6, 13, 16

[12] Stephanie L Hyland, Eth Zurich, Cristbal Esteban, and Gunnar Rätsch ETH Zurich. REAL-VALUED (MEDICAL) TIME SERIES GENERA-TION WITH RECURRENT CONDITIONAL GANS. Technical report. URL: https://arxiv.org/pdf/1706.02633.pdf. pages 2, 3

[13] Umar Iqbal, Peter Snyder, Shitong Zhu, Benjamin Livshits, Zhiyun Qian, and Zubair Shafiq. ADGRAPH: A Graph-Based Approach to Ad and Tracker Blocking. Technical report. URL: https://arxiv.org/pdf/1805.09155.pdf. pages 1

[14] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. 12 2014. URL: http://arxiv.org/abs/1412.6980. pages 12

[15] Mohammad Malekzadeh, Richard G Clegg, Andrea Cavallaro, and Hamed Haddadi. Mobile Sensor Data Anonymization. 10(19), 2019. URL: https://doi.org/10.1145/3302505.3310068, doi:10.1145/3302505.3310068. pages 9, 11

[16] Mohammad Malekzadeh, Richard G. Clegg, and Hamed Haddadi. Replacement AutoEncoder: A Privacy-Preserving Algorithm for Sensory Data Analysis. In *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 165–176. IEEE, 4 2018. URL: https://ieeexplore.ieee.org/document/8366986/, doi:10.1109/IoTDI.2018.00025. pages 11

[17] Olof Mogren. C-RNN-GAN: Continuous recurrent neural networks with adversarial training. Technical report. URL: https://github.com/olofmogren/c-rnn-gan. pages 3

[18] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic Image Synthesis with Spatially-Adaptive Normalization. Technical report. URL: https://arxiv.org/pdf/1903.07291.pdf. pages 3

[19] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. 11 2015. URL: http://arxiv.org/abs/1511.06434. pages 3

[20] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. A Hybrid Convolutional Variational Autoencoder for Text Generation. Technical report. URL: https://arxiv.org/pdf/1702.02390.pdf. pages 3

[21] Egor Zakharov, Aliaksandra Shysheya, Egor Burkov, and Victor Lempitsky. Few-Shot Adversarial Learning of Realistic Neural Talking Head Models. Technical report. URL: https://arxiv.org/pdf/1905.08233.pdf. pages 3

[22] Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A Efros, and Berkeley Ai Research. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks Monet Photos. Technical report. URL: `https://arxiv.org/pdf/1703.10593.pdf`. pages 3