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 training data itself was taken from the `MotionSense` dataset, and the features used in the models were the value of the three orthogonal accelerometer measurements. 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 `MotionSense`, an accuracy score of 74 % is achieved.

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*. Founded in 2015, *Brave* are a new, privacy-focused web browser which is seeking to introduce a better web browsing ecosystem through, amongst other techniques, 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 [XX adgraph paper]. It also addresses the issue of web trackers, which are a major privacy concern, by providing automatic tracker blocking and fingerprinting prevention.

Of particular interest to this project, however, is their proposed advertising model. In the current advertising ecosystem, revenue from advertising is generally collected by the websites on which they are hosted. 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[XX], 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 [X] 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`[X], 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 is possible to 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 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 which exhibits sequentiality. A particularly interest use case, where much work is being done [X], is for privacy protection matters. It would for example be possible, in principle, to generate synthetic medical data, which would otherwise be difficult to obtain.

Chapter 2

Background of models

In this Chapter, several state-of-the art models are formally introduced, and their underlying theoretical frameworks are used to justify which model is used for the work done in the project.

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[XX] and text[XX] 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 adverisal networks(GANs).

However, much of the work done with GANs, has focused on image generation. These models are based on deep convolutional neural networks which are now been well established (10) (12). There has however, been limited work done on the generation of one-dimensional data (9) (5) using GANs. Though generating one-dimensional data may appear to be a trivial problem once the two-dimensional case is successful, there are caveats which need to be dealt with, when attempting to generate one-dimensional data. The Deep Convolutional GAN (DC-GAN), upon which many image based GANs are built upon (11), illustrates this well. DC-GANs contain multiple two-dimensional convolution layers, which allows the neural network to contextualise and localise information using information along two axes. This would not be possible for the one-dimensional case since convolutional kernels would, by definition, be restricted to one dimension.

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 using what it has learned from training on genuine data. (4)

Through the combined training of these two networks, it is possible to obtain a well trained generative model, which should theoretically be capable of producing

data which is indistinguishable from the training data.

Figure 2.1 illustrates the training process well. Given a large enough training set, it is possible to learn the statistics of the set of real samples well enough to be able to distinguish between samples belonging to the training set, and those that do not. In this form, the problem can be thought of as a binary classification problem, where the discriminator is trained to accept the the training data as genuine, and the generated data as fake. Meanwhile, the generator is trained to fool the discriminator, by attempting to get the discriminator to mislabel samples it produces as genuine.

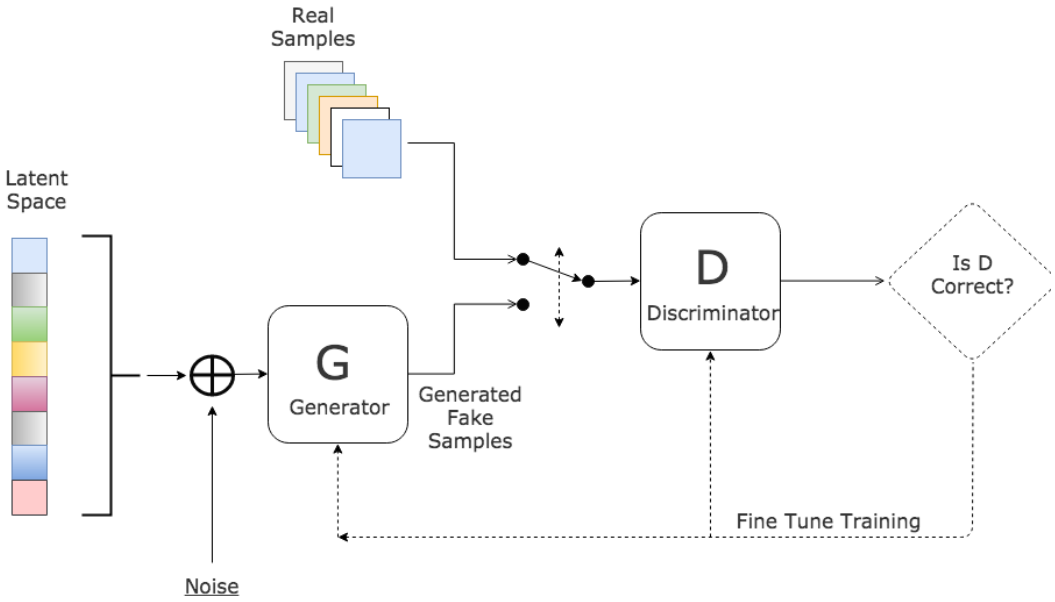

Figure 2.1: Schematic of the fundamental architecture of a GAN(1)

At the discriminator stage, the neural network is performing a binary classification task, hence one appropriate loss function for training is the binary cross entropy function (4), 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 labels of the training and generated data respectively.

In order to maximise the performance of the discriminator, it trained to accept the real samples, and reject the generated samples.

The generator, however, is trained to maximise the score given to the generated samples, i.e. maximise $D(G(z))$, where D and G are the discriminator and generator functions respectively, and z is the latent space from which G maps into the sample space.

Following Goodfellow's treatment (4), one can reach a form for the value function of a mini-max game as

$$\min_{\boldsymbol{G}} \max_{\boldsymbol{D}} V(\boldsymbol{G}, \boldsymbol{D}) = log\boldsymbol{D}(\boldsymbol{x})) + log(1 - \boldsymbol{D}(\boldsymbol{G}(\boldsymbol{z}))), \tag{2.2}$$

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 section, the way in which the programme was developed is described 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 way in which they can be achieved is determined (3). 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' (3).

The 'Waterfall method', which is 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 allows the flexibility to build multiple 'products' which was deemed to be sufficiently well suited for this project, as products can be developed in a cyclical manner with increasing complexity.

3.2 Architectural Design

An object-oriented design was selected for the project since this lends itself well to modularity that object oriented design allows. The modularity allows the production of a program which is easily maintanable and to which further functionality can be added. Furthermore, for machine learning projects such as this, having an object-oriented design is useful, since hyperparameters can be stored as attributes of objects which can easily be initialised.

The packages have been organised into a trainer module and gan models. The trainer module contains the procedures in which the training of the GAN takes place, whilst the models themselves are stored as objects of the gan modules which contain the details of the architechtures of the discriminator (or critic, in the case of the Wasserstein variations) and the generator of the GANs.

The `DataPreparation` class contains routines used for the extraction and re-structuring 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.

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). The training time of the models with, and without, GPUs are quantified in the results section; however, as expected, GPUs gave a significant increase in time efficiency. This is due to the capability of the GPU to parallelise matrix multiplication calculations, which is what fundamentally core to the training of neural networks. Furthermore, Colab offers version control by allowing access to revision history.

3.4 Training data

3.4.1 Sources of data

The sensory data used for training the models is taken from the `MotionSense` dataset [XX]. 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 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`, 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.

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 [XX][XX][XX]; however, here 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, standardisation was used, following the original Malekzadeh implementation (8). When training the generative models, however, normalisation was used. Tanh blah nlah blah.

The data is normalized separately across each of the three channels of sensory data, to produce a normalised set of data between -1 and 1. This was chosen as it is recommended for GAN training due to the utilisation of a hyperbolic tangent activation function and the range of tanh is between -1 and 1.

3.4.3 Exploratory Data Analysis

The data used for this experiment was first explored to cement understanding of its features, usability and limitations. An example of the data, taken from the XX experiment where XX is being performed is shown in Figure XX. The models here have been trained on accelerometer data only. This was to show the feasibility of the model and to reduce simulation time. Furthermore, the accelerometer data has been used in the paper [XX] hence it is known to produce results.

The accelerometer data in three directions is shown in Figure. We can observe a relationship between the three axes of motion, as well as the combined effect of the magnitude.

The aim of the project is to produce datasets such as these which can mimic this behaviour.

Further investigation shows the variability of the patterns. Although not as trivial to notice as for image data, where our human eyes are well capable of pattern recognition for everyday objects, we can still see distinct types of motion 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 (7) 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 over 95 % 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 cruicial 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 overtrained. The `Trainer` module has been designed such that the optimisers for both the discriminator (or critic) and generator are stored as attributes of instantiations 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` optmiser was used with the learning rate of 0.001, as suggested in the Adam paper (6), 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 (2) 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 .

The optimiser for the for the Wasserstein GAN implementation was the `RMSProp`. This was selected based on the recommendation of the original Wasserstein paper. It is applicable since XXX

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.

Thu, changing how the training data is prepared, using the `DataPreparation` 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 Standard GAN

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.

The discriminator network, takes as its input a set of data and label, and returns a score in the range [0,1], indicating how likely it is to have been correctly identified as either real or fake.

The convolutional and pooling layers are one-dimensional; they only act in the axis of a single stream of sensory data. This is to ensure there is no information transfer between the streams when considering the sequentiality of the data along each axis individually.

3.8 Training process

The instantiations of the model classes contains the the discriminator and generator networks as attributes. These instantiations are passes through the Trainer class which contains methods for training the different types of 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 (4).

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 (2). 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 incoorprated 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

The most successful of the models that was developed was a deep convolution generative adversarial network, the architecture of which was inspired by the classification model developed for the `MotionSense` dataset by its authors. Although a conditional GAN was developed during this project, data from a particular class could be isolated in the training set, and then used to train the generator with which new samples of that class can generated. Thi

4.1 Monitoring the Results

Monitoring the results of the performances of the networks proved to be useful. This is because, often, if it is observed that learning has stopped, there is not point in continuing with the simulation. Once networks were tuned however, the plots observed match the theoretical predictions.

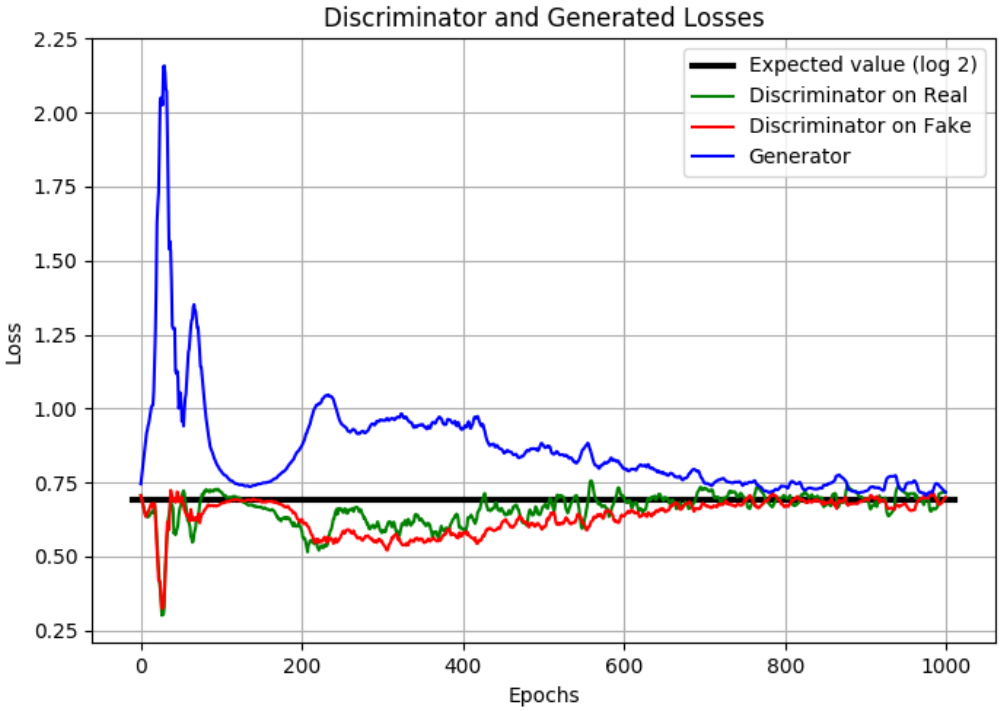

Figure 4.1: Losses per epoch

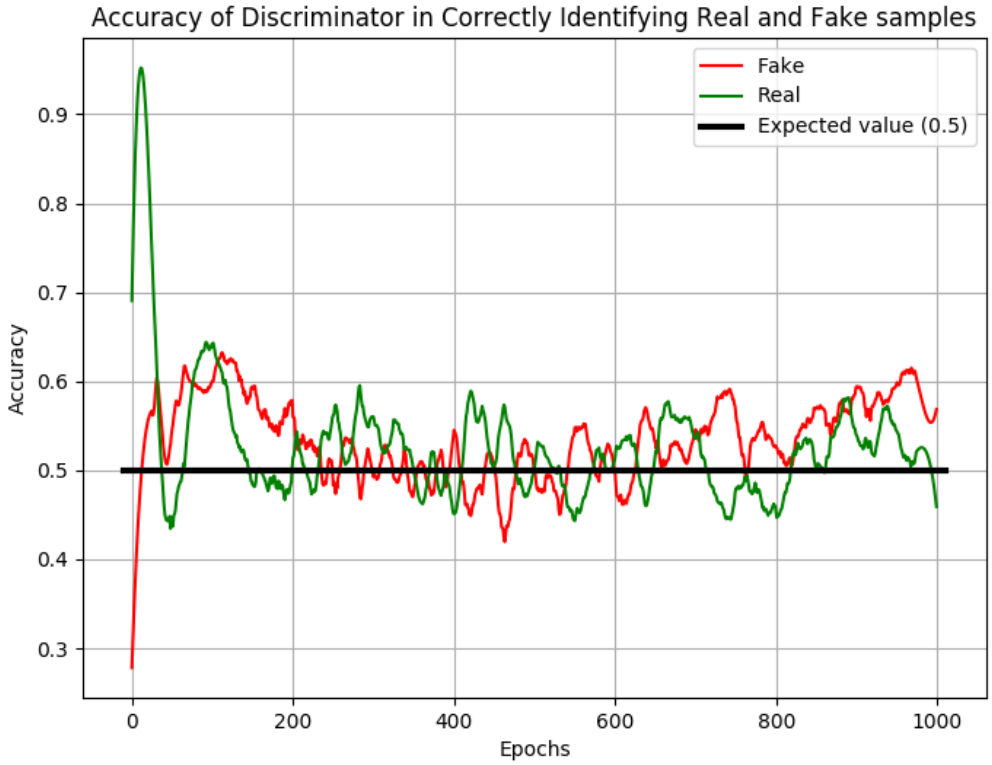

Figure 4.2: Accuracy of the discriminator per epoch

4.2 Visual Checking

The first stage in assessing the quality of the results, is by simply observing a plot of the data. Examples of a generated sequence and a sample from the training set are given in Figure 5.1. Whilst the Figure only shows one example of the generated sequences, they are representative of the results that can be produced by this model once convergence occurs.

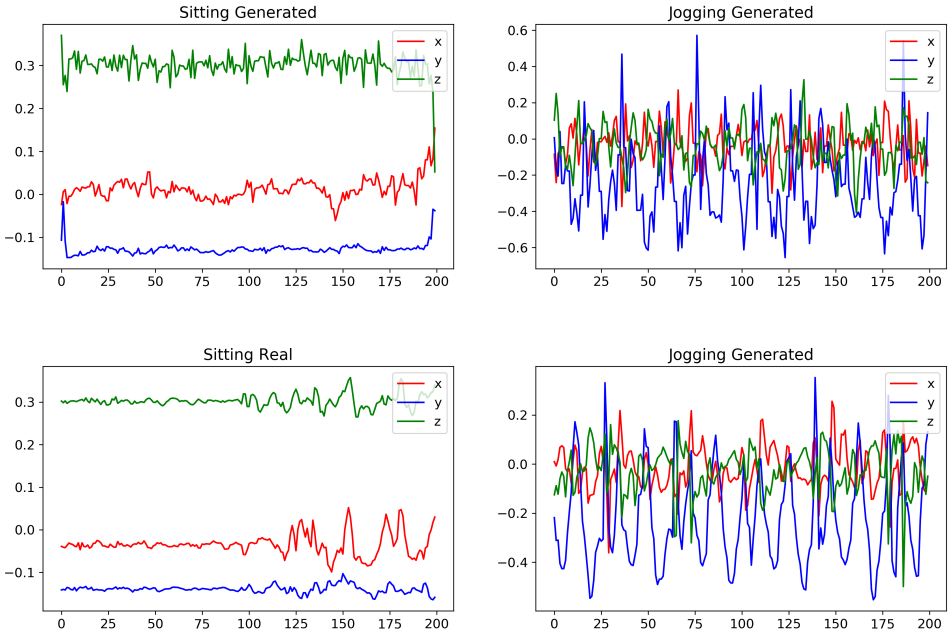

Figure 4.3: Caption

Chapter 5

Discussion and Conclusions

5.1 Difficulties

Compare with GANs for image generation. Human eye is good (or trained, joke here) at reconising patterns in MNIST or CIFAR 10 as these are shapes and patterns we see in every day life. We are good at distinguishing real and fake horses but not time series sensory data.

Training GANs are unstable

5.2 What went well

5.3 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 architechtures 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 arbritrarily. This could potentially be done by training another neural network to join existing sequences together, or by constructing the GAN such that

Bibliography

[1] Generative Adversarial Networks . URL: https://www.kdnuggets.com/2017/01/generative-adversarial-networks-hot-topic-machine-learning.html. pages 4

[2] soumith/ganhacks: starter from "How to Train a GAN?" at NIPS2016. URL: https://github.com/soumith/ganhacks#authors. pages 10, 12

[3] 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 6

[4] 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 3, 4, 11

[5] 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 3

[6] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. 12 2014. URL: http://arxiv.org/abs/1412.6980. pages 10

[7] 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

[8] 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 9

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

[10] 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

[11] 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

[12] 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