Generating realistic human sensor data using Conditional Generative Adversarial Networks

Research Paper Business Analytics
Vrije Universiteit Amsterdam, The Netherlands

Author: Maarten Kleinrensink
Supervisors: Dr. Mark Hoogendoorn and Ali el Hassouni
April 2019

Abstract. Conditional Generative Adversarial Networks (CGAN) are shown to be useful for generating images. However, the use of CGANs for human sensor data, is not investigated. It is expensive to gather labeled data and it is privacy-sensitive to use this data, therefore it is very desirable to be able to generate realistic looking data in order to replace the collected data by the generated data, to avoid these privacy issues. The data is needed to be able to train a model to determine what activity someone is performing at that moment. This Activity Classifier could be implemented in a simulation environment like in [1]. It has been shown that Generative Adversarial Networks (GAN) can be created to generate realistic human sensor data in paper [1]. In this paper a CGAN is developed, which can generate various types of sensor data that is related to activities someone is performing. To determine the performance of the generated data, an Activity Classifier is created. However, the generated data is not reliable to use, because the data is visually significantly different than the actual sensor data and the activity classifier is only able to classify the correct label to three of the six activities. Some methods and advice will be given on how a CGAN, that generates realistic human sensor data for different conditions can be developed.

1 Introduction

Nowadays many people carry one or multiple smart devices on them during the day, like smart phones, smart watches etcetera. These smart devices can measure movement with multiple sensors. If Machine Learning is used to predict what activity someone is performing at that moment, we need labeled sensor data. The collected data can be useful to improve the physical condition of the individual. This can be done by an intervention generated by a simulation environment, like described in [1]. However, to generate useful recommendations, the activities that the individual has performed during the day must be known. A large amount of sensor data is needed to train a model that accurately predicts what activity someone is performing given a sequence of data points.

The data that is used to train the model, must be labeled with the activity someone is performing at that moment. The problem is that it is very expensive to gather data of individuals that keep track of their activities while using a smart device that collects the data. Furthermore, the data that is collected, is privacy-sensitive. To avoid privacy issues with the sensor data, extra data can be generated, in order to replace the sensitive collected data with the generated data. It is important that this data sequences are not just copies of previous collected data, but it should be data that includes variation and noise, because each individual is unique and will have different sensor data related to an activity. The main reason of this research is based on paper [1]. This research is an extension of that paper, a brief summary of this paper will be given in section 2.

The main focus of this paper will be to describe a method to generate more labeled data that includes noise and is reliable to use for a predictive model. The method that will be used in this paper is a Conditional Generative Adversarial Networks (CGAN) [2], which is an extension of Generative Adversarial Networks(GAN) [3]. This makes sure that not multiple GANs should be created to generate labeled data of multiple activities, but just one CGAN can do the job. The use of CGANs are shown to be successful for generating images in, for example, [4]. However, the use of CGANs for human sensor data, is not investigated. The main research question of this paper is:

Is it possible to generate realistic human sensor data using a CGAN?

An Activity Classifier is created to help evaluate the generated sequences of the CGAN. Furthermore, the visual looks of the generated data sequence is compared to the real data, the Discriminator- and Adversarial Loss and the accuracy of the Discriminator determine the performance of the CGAN. Where the outcome of the Activity Classifier is most leading. The CGAN and the Activity Classifier will be created using LSTM networks [5], [6]. Another topic that will be discussed in this paper is what structure the CGAN should have to create a reliable model.

This paper is organized as follows. First, related work will be discussed in section 2. A general description and theoretical background of Generative Adversarial Networks and Conditional Generative Adversarial Networks will be given in section 3. The data that is used to create the Conditional Generative Adversarial Networks and the Activity Classifier is described in section 4. Then the setup and performance of the Activity Classifier is described in section 5. Followed by the experiments conducted, the setup and the results of the CGAN in section 6.2. Also, some further research is done on the dataset including demographics of the individual who is linked to the sensor data. This will be described in section 7. Finally, a conclusion will be given in section 8.

2 Related work

The paper [1] was the main reason for this research. They succeeded to create a GAN that could generate realistic human sensor data. The data needed to create the GAN of [1], is comparable to the dataset used in this paper. The use of GANs is proposed so that behavioral models that mimic human behaviour can be created. This is an approach to take simulation environments for e-Health to the next level. In [1] an existing simulation environment is used which is extended and made more mature. The focus of the simulator is on a health setting where users get interventions to adapt their activities. They make use of an Activity Classifier to evaluate the performance of the GANs. This classifier is able to tell what activity the raw sensor data is from with an overall accuracy of 97.33% on the test set. The GAN is able to generate data where the classifier is able to classify, on average over 500 iterations, up to 97% correctly. The GAN did not work for every activity. For example, the data generated for the activity standing was classified correctly 7% of the times.

[2] was the first to investigate the use of a CGAN. In a previous research they have shown that a GAN can be used as an alternative framework to train generative models. This paper goes deeper into that research and shows that it is possible to create a CGAN which is able to generate images. A CGAN can generate images for multiple conditions, while a GAN just generates images of one condition. So just one CGAN can replace multiple GANs. However, the results of the CGAN are outperformed by the GAN. The authors of [2] are convinced that the results of the CGAN should match or exceed the results of a GAN if further research is done for the parameters and architecture of the CGAN. Combining the findings of [1] and [2] gave the impression that it would be possible to create a CGAN for realistic human sensor data.

In [7], a comparable investigation is performed. Namely, the option to use a CGAN for convolutional face generation. The CGAN that is build is able to generate images of faces. They filtered out the face attributes which have clear visual effects in the images so that the model could be trained more easily. This makes sure that the CGAN is not overfitting and it avoids that the CGAN does not see the unclear visual effects as random noise. [8] investigated the option to use a CGAN for spoken language identification. This paper proposes a new structure for a CGAN. A comparable structure is shown in figure 10b. This CGAN has a Discriminator that not only determines if the data is real or not, but also indicates which label should correspond to the data. The i-vector data based on NIST 2015 language recognition i-vector machine learning challenge was used for examination with identification error rate as the evaluation criterion. The conclusion of the research is that the algortihm proposed is outperformed by other Deep Neural Networks, However, the proposed CGAN structure is working well.

Overall, most of the research done concerning Conditional Generative Adversarial Networks are mainly focused on generating images. Like [9] and [4], they created a CGAN which translates edges of a variety of images to high-resolution natural photos. In [10] a CGAN is presented which can dehaze unclear images. and [11] created a CGAN that changes the image of an existing face into the face in the wanted age category. In all the papers discussed in this section that created a CGAN, the CGAN is build out of many large hidden layers. Also a large number of epochs are needed to have a well working CGAN. This will not be doable for the research done in this paper while the computational power available is not sufficient for such complex models, because this will take a too long time to train the CGAN and thus very time consuming to test multiple parameter settings.

3 Model Description

The algorithm used is a Conditional Generative Adversarial Networks, which is an extension of a regular Generative Adversarial Networks. To have a better understanding of a CGAN, a regular GAN will be explained first. After this, the CGAN will be discussed.

3.1 Generative Adversarial Networks

A method to generate more labeled sensor data is to use Generative Adversarial Networks. The main focus of the GAN is to create data from random noise. Where GANs combines two deep learning models, The Generator (G) and Discriminator (D) [3]. In short, a Generator generates data, which should look as real as possible. This data is generated from random noise. A Discriminator determines the quality of the generated data by trying to distinguish real and generated data, which is shown in figure 1. In this figure Z indicates the random noise, X_{real}, a real data sequence and X_{fake} the generated data sequence by the Generator.

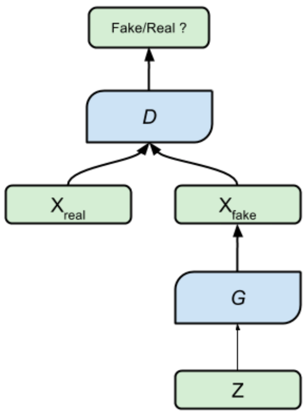


Fig. 1: Generative Adversary Network [12]

If the Discriminator cannot determine what is real data or generated data and the generated data has approximately equal statistics as the real data, like mean, variance etcetera, then it is a possibility to replace the original data by the generated data.
The overall objective function of a GAN is given in equation (1).

$$\min_G \max_D [\mathbb{E}_{x \sim p_{data}} \log(D(x)) + \mathbb{E}_{z \sim p(z)} \log(1 - D(G(z)))] \tag{1}$$

Where $D(x)$ is the Discriminator output for real data x and $D(G(z))$ is the Discriminator output for generated data $G(z)$. The aim of the Discriminator is to maximize the objective such that $D(x)$ is close to 1 and $D(G(z))$ close to 0, where 1 indicates that it is real data and 0 fake data. The Generator wants to achieve the opposite. The Generator wants to minimize the objective such that $D(G(z))$ is close to 1. In other words, the Generator wants to trick the Discriminator in thinking that $G(z)$ is real data, while the Discriminator wants to identify real and fake data as correct as possible. The goal of the GAN is to generate a realistic output by alternating between gradient ascent on Discriminator given by equation (2) and gradient descent on Generator given by equation (3).

$$\max_D [\mathbb{E}_{x\sim p_{data}} \log(D(x)) + \mathbb{E}_{z\sim p(z)} \log(1 - D(G(z)))] \tag{2}$$

$$\min_G \mathbb{E}_{x\sim p(z)} \log(1 - D(G(z))) \tag{3}$$

```
for number of training iterations do
    for k steps do
        – Sample minibatch of m noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from noise prior $p_g(z)$.
        – Sample minibatch of m examples $\{x^{(1)}, ..., x^{(m)}\}$ from data generating
          distribution $p_{data}(x)$.
        – Update the Discriminator by ascending its stochastic gradient:
              $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [\log(D(x^{(i)})) + \log(1 - D(G(z^{(i)})))]$   (4)
    end
    – Sample minibatch of m noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from noise prior $p_g(z)$.
    – Update the Generator by descending its stochastic gradient:
          $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [\log(1 - D(G(z^{(i)})))]$   (5)
end
```

The gradient-based updates can use any standard gradient-based learning rule.

Algorithm 1: GAN training algorithm [3]

To obtain the goals described in equations 2 and 3, the algorithm described in algorithm 1 is used. This algorithm shows that per iteration the Discriminator is trained k steps with random noise and real data. Per step, the Discriminator is updated by ascending its stochastic gradient using the random noise and real data. After the k steps the Generator is updated by descending its stochastic gradient using random noise. [3]

3.2 Conditional Generative Adversarial Networks

A GAN can be extended to a CGAN when an extra condition is added as input for the Generator and Discriminator. This condition will be an extra input when training the Generator, as well as the Discriminator. The objective function of the CGAN is given in formula (6).

$$\min_G \max_D [\mathbb{E}_{x \sim p_{data}} \log(D(x|y)) + \mathbb{E}_{z \sim p(z)} \log(1 - D(G(z|y)))] \tag{6}$$

For the Generator the input noise z and the label y are combined into a hidden layer to generate data of label y. The Discriminator tries to identify, like a GAN, if the data x with label y is real data or not. An illustrative example of a CGAN is shown in figure 2. The outcome of the Discriminator is an identifier that identifies if the data with the corresponding label is real or not [2].

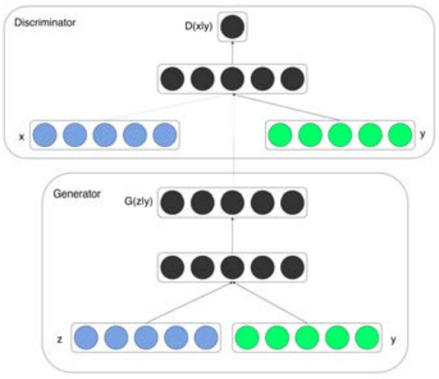


Fig. 2: Conditional Generative Adversary Network [2]

Long-Short term memory networks
Long-Short term memory (LSTM) networks belong to the Recurrent Neural Network (RNN) family. An illustration of a RNN is given in figure 3 with A the cell, x_t the input and h_t the output of the cell at time t. A RNN is a sequence of multiple Neural Networks (NN), where the output of the previous NN is taken into account for the next NN. For the sequential accelerometer data described in section 4, it will be useful to use LSTM networks. LSTM networks are an extension of RNN where it is able to learn long-term dependencies [5]. LSTM networks are useful to learn from experience to classify, process and predict time series. Even if there is a (very) long time of unknown size between crucial and relevant events [6]. This motivated to use LSTM networks for both the Generator

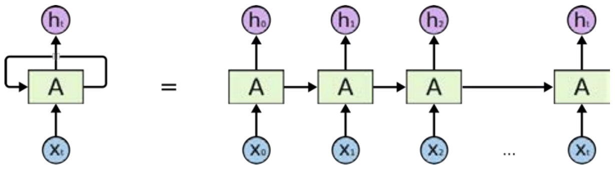


Fig. 3: Recurrent Neural Network [5]

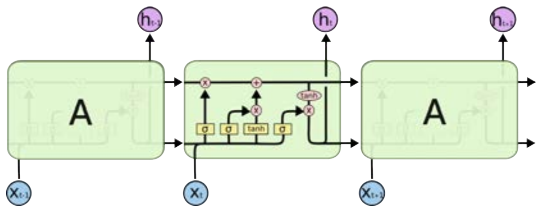


Fig. 4: Illustrative example of LSTM cells [5]

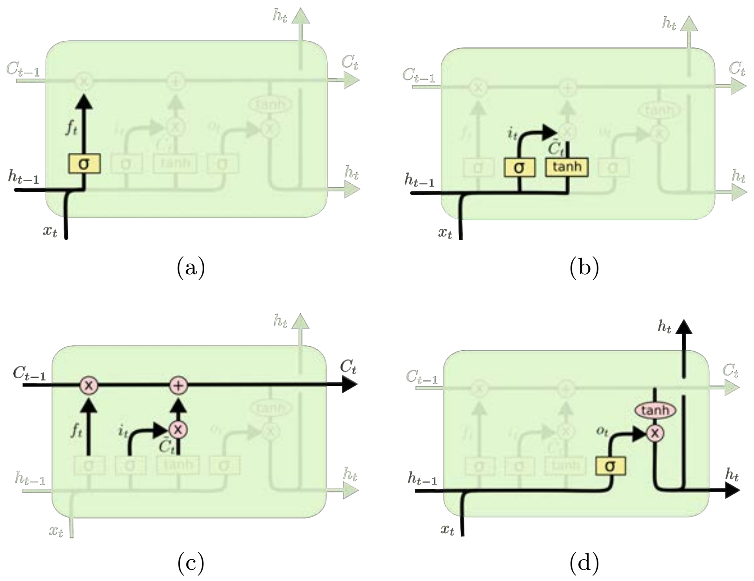


Fig. 5: Layers of LSTM cell [5]

and the Discriminator. LSTM networks are also used for the Activity Classifier described in section 5.

Figure 4 shows an example of the layers of a LSTM cell. The layers of a LSTM cell, step by step, are given in figure 5. The flow of information between time point $t-1$ and t is called the cell state C_t. The LSTM can add or remove information from the cell state handled by gates, which are carefully regulated [5]. Figure 5(a) shows the forget gate layer, Which is the first step of a LSTM. It decides how much of the information must be thrown away from the cell state looking at h_{t-1}, the output of the previous layer and x_t, the input of the current layer. The value f_t of the forget layer is calculated using equation (7) with W_f the recurrent weights of gate f and b_f the bias weights for gate f. This is a sigmoid funtion, which will result in a value between 0 and 1, where a value of 1 indicates that all the information will be kept and nothing will be thrown away. If the value becomes 0, all the information will be thrown away.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{7}$$

The next step of the LSTM is to decide what information should be saved in the cell state. This will be done in two phases, which is shown in figure 5(b). First, the input gate layer determines which values should be updated. The output of this step is calculated using the sigmoid function (8). After this, the tanh layer creates information that could be added to the cell state. This is calculated using equation (9). The values i_t and $\tilde{C}_t$ are used to update the old cell state C_{t-1} and create a new cell state C_t. Equation (10) shows how this is done. The old cell state is multiplied by f_t, to throw away all the information that is not needed from the previous cell state. The new candidate values are calculated by multiplying i_t and $\tilde{C}_t$ to scale each state value. The illustrative representation of this step is shown in figure 5(c).

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{8}$$

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{9}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{10}$$

The output of the current layer will be based on the cell state C_t. What parts of the cell state will be used is determined by sigmoid function (11). To make sure the values of the cell state are between -1 and 1, the cell state is run through a tanh layer. This is multiplied by o_t in order to only get the output h_t that is determined to output. These steps are shown in figure 5(d).

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{11}$$

$$h_t = o_t * tanh(C_t) \tag{12}$$

4 Data

For this paper data from multiple people is used, released by the Wireless Sensor Data Mining (WISDM) Lab [13]. The sensor data is generated by the accelerometer and this data has labels of multiple activities. This data is real world data where individuals mark what activity they are performing at that specific moment. The raw data consists of 2,980,765 data points and demographics data of 563 participants is noted. In this dataset, 6 different activities are present. The number of datapoints per activity are given in table 1. As this table shows, some activities have (very) little data available in comparison to other activities. Especially walking the stairs has limited datapoints.

Activity	**Data points**
Jogging	435,238
Lying down	275,966
Sitting	663,142
Stairs	56,895
Standing	288,871
Walking	1,250,427

Table 1: Number of data points per activity

Data of the x-, y- and z-accelerometer have a very different pattern per activity. Examples of these different patterns are shown in figures 6 and 7. The data generated by the accelerometer is saved with a rate of 20 HZ, which indicates that every 50 ms a datapoint is saved for the values of the accelerometer. The plots given in figures 6 and 7 consist of 160 datapoints, which is 8 seconds of data per activity. The assumption is that performing an activity, will last for at least 8 seconds. The x-, y- and z-accelerometer data is shown in the colors blue, green and red respectively.

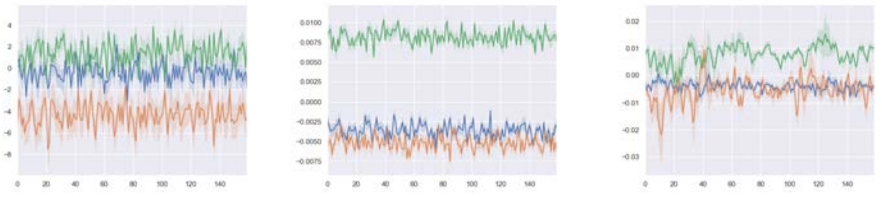


Fig. 6: x-, y-, and z-accelerometer data of jogging, lying down and sitting

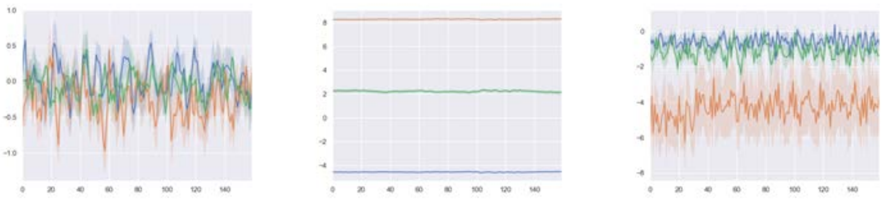


Fig. 7: x-, y-, and z-accelerometer data of stairs, standing and walking

Figures 6 and 7 clearly indicate the difference between each activity. Interesting to see is, while standing, the accelerometer data has relatively high values compared to the activities lying down and sitting. These two activities have values from around 0, where standing has values from approximately -5, 2 and 8, While all three activities are passive. Another thing that stands out is that the data of standing in this example, is almost constant. However, the values of lying down and sitting are so small, that this data for these examples are almost constant as well. When walking, Jogging and walking the stairs, these fluctuations are more present.

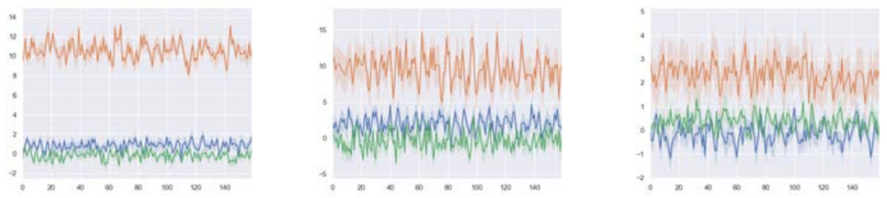


Fig. 8: x-, y-, and z-accelerometer data of stairs, jogging and walking

Figure 8 shows that the accelerometer data for the stairs, standing and walking can vary significantly in comparison to the x-, y- and z-accelerometer data given in figures 6 and 7. An explanation for this is that every person has a different age, height and weight, which influences the speed of walking, jogging or walking the stairs. Where sitting, lying down and standing does not fluctuate that much per type person. Data sequences of one person differs as well for these activities, this can be caused by, for example, performing activities at different speed or having a different stepsize while performing the same activity.

5 Activity Classifier

To evaluate the performance of the CGAN, an Activity Classifier is created. This classifier tries to determine the activity of the generated data. In other words, if the generated data is comparable to the actual data of the activity. The optimal performance is when the generated data of all the activities can be classified with the correct activity. The setup and performance of the activity classifier are described below.

5.1 Setup

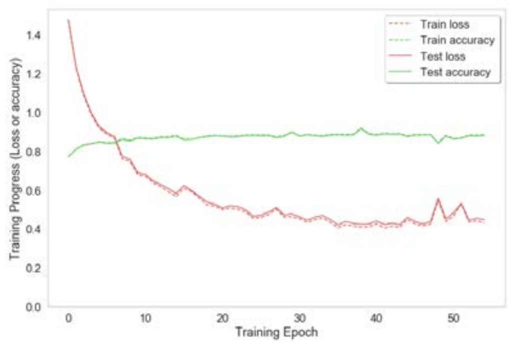
the Activity Classifier is trained on the data described in section 4. This contains real life x-, y- and z-accelerometer data of six activities. Segments of 8 seconds are created in the data, which are 160 datapoints. These segments were created with a shifting window of 1.5 second (30 datapoints). A shifting window of 30 datapoints is chosen because this had optimal results of the accuracy and loss, where shifting windows between 10 and 60 datapoints are tested. Each segment has a label that occurred in that time window. If more activities are present in that time window, the most occurring activity is chosen. The segments created are divided into a train and test set. Where 80% of the segments are used for training and the remaining 20% for testing the classifier.

The structure of the classifier is an LSTM network with 2 hidden layers. Where each hidden layer has 64 hidden units and a ReLu activation with a forget bias of 1. A softmax activation was used for the output layer with a forget bias of 1. The Adam optimizer was used with a learning rate of 0.0022. The cost function was a Softmax cross entropy including a logits with L2 regularization of 0.0018. The batch size during training is 1024. The structure and parameter settings were based on the paper [1] and [14]. Multiple settings for the number of hidden units per LSTM layer are tested, the batch size during training, activation functions (tanh, sigmoid, softmax and ReLu) and number of hidden LSTM layers. The settings described above were performing the best based on the accuracy and loss of the classifier. The performance is described in section 5.2.

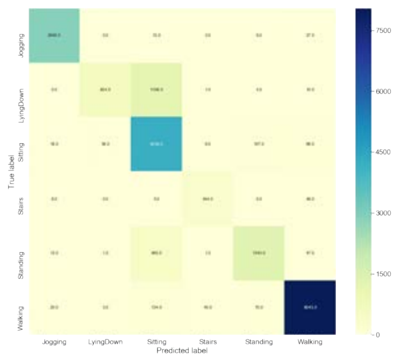
For a well working classifier, with a low computational time, it is possible to chose 32 hidden units per hidden LSTM layer and a batch size during training of 512, where the rest of the settings can remain the same. However, as data preparation segments of 160 datapoints should be created with a shifting window of 20 datapoints. The performance was just slightly lower (0.3% less accurate overall), but was made in a fraction of the time.

5.2 Performance

The performance of the activity classifier are shown in figure 9. Where figure 9a shows the accuracy and loss on the train and test set per training epoch. The classifier had an accuracy of 88.0% on the test set and a loss of 0.427. The loss and accuracy of the train set do not differ a lot from the accuracy and loss on the test set. However, the accuracy on the training set is slightly higher and the loss on the training set is slightly lower than the test set. Figure 9b shows the confusion matrix based on the predicted label and the actual label of the test set. This shows that the classifier is working well for most activities, however, for the activities Lying Down and Standing, it is performing poorly. The activities Lying Down and Standing are misclassified most often as Sitting, which are all passive activities. The more active activities can be classified more accurate. However, walking the stairs has a lower performance than the overall performance. Walking the stairs is misclassified most often as Walking. Table 2 shows the percentage correctly predicted per activity.



(a) Accuracy and loss



(b) Confusion matrix

Fig. 9: Performance of Activity Classifier

Activity	
Jogging	98.3%
Lying down	33.9%
Sitting	94.7%
Stairs	85.3%
Standing	70.1%
Walking	96.7%

Table 2: Percentage correctly predicted per activity on test set

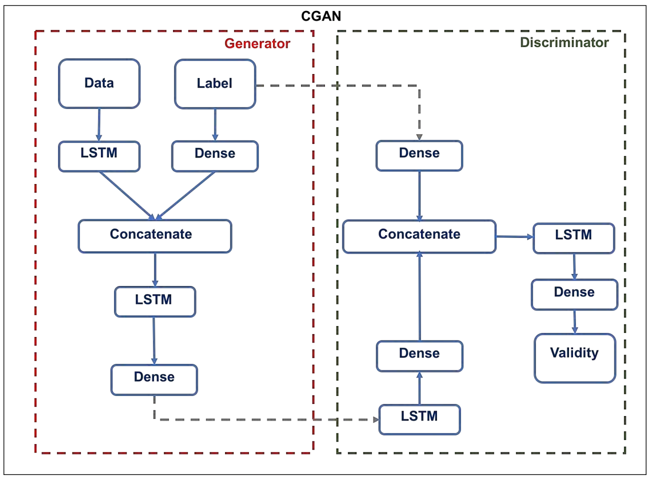
6 Experiments

CGANs shown in figure 10 are a general representation of the models tested. These models are tested based on the theoretical shape of a CGAN described in [2] and the new proposed structure described in [8] and other research done. The input of the Generator must be random noise. The output of the Generator including the label it should represent will be the input of the Discriminator as well as samples of the real data of that label, so that the Discriminator can try to make a distinguish between real and fake data. CGAN version 2 shown in figure 10b, shows a significant difference with the theoretical shape of a CGAN described in [2] and shown in figure 2, Which is more like the CGAN version 1 shown in figure 10a. In CGAN version 2, the output of the Discriminator is a label and a validation and not only a validation. This structure is described in [8]. The original concept of a CGAN as in figure 2 and like figure 10a, is to validate if the data is real or not and not also have a label as output. However, after some trial and error and some research, it is advised to create a CGAN comparable to version 2. The performance of the CGAN is based on the visual looks of the generated data sequence in comparison to the real data, the Discriminator- and Adversarial Loss, the accuracy of the Discriminator and most leading, if the Activity Classifier is able to classify the generated sequence correctly.

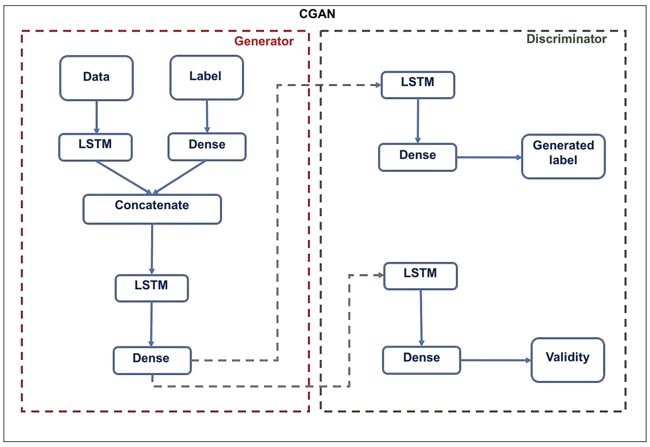
The Dense and LSTM functions in the CGANs are tested with many settings for the activation and number of hidden nodes to find optimal parameters for the Dense and LSTM functions. Activations tested are tanh, sigmoid, softmax and ReLu. As seen in figures 10a and 10b, there are not many layers in the Generator and Discriminator. However, these models are tested with more layers as well. Nevertheless, this had a lower performance as well as a long computational time. This could be caused by the size of the layers, since these layers could not consist of many hidden nodes where this would increase the computational time too much. In research done, as well in the articles described in section 2, most of the CGANs had many layers with many hidden nodes, so the expectation is that in general this would increase the performance of the model. Data preparations tested are: Measured accelerometer data, transformed data (all data between so 0 and 1 and all data between -1 and 1) and measured accelerometer data data of three activities. The measured accelerometer data had the best performance and thus will be used for the CGAN. The label that indicates the activity, must be transformed to a numerical value. This can be done by giving each activity an integer value or to give each activity a string of zeros and a one at the location to indicate what activity it is. The latter is called one hot encoding. For example, activity 5 will be [0,0,0,0,1,0] if one hot encoding is used. The best performance of the CGAN was when the label was converted into a one hot encoded string. The results represented in section 6.2 are created with a one hot encoding for the labels.

Per iteration the CGAN is trained per activity. After all activites, the iteration is complete. So the CGAN has 6 updates per iteration. After every iteration, the activities are shuffled, so that the CGAN cannot train for a pattern in which the activities occur but solely trains on the data corresponding to the activities.

Optimizers tested for the Generator, Discriminator and CGAN are the Adam, RMSProp and SGD optimizer. The best performing combinations are given in section 6.1



(a) CGAN version 1



(b) CGAN version 2

Fig. 10: CGAN structures tested

6.1 Conditional Generative Adversarial Networks Setup

Using the data described in section 4, a CGAN is created. The CGAN is trained to generate x-, y- and z-accelerometer data for multiple activities. As a first step, the dataset is divided into segments of 160 datapoints, without overlap so that these can be used for training the CGAN. The Generator is trained with 24 segments of real sensory data of an activity and 1 batch of generated data of the same activity. Taking more batches or more data points, had a negative effect on the performance of the CGAN. The CGAN is trained using Binary cross entropy and Categorical cross entropy as Loss functions and the Adam optimizer with a learning rate of 0.01 with a learning rate decay of 1e-10, β_1 of 0.20 and β_2 of 0.25. Accuracy is used as the performance metric of the CGAN. The inspiration for these settings came from [1] and for the structure from [8]. A visual representation of the CGAN structure used is given in figure 10b, combining the Generator and the Discriminator.

Generator
For the Generator, there is noise of shape (160,3) as input and a label of shape (1,6) as input. The noise goes through one hidden layer. This is a LSTM layer with 32 hidden units. The label goes through a fully connected layer of 32 hidden neurons with a softmax function as activation. Then these outputs are merged with a concatenate function in the third dimension. So the output of the LSTM layer (160,3,32) and the fully connected layer (1,6,32) are merged in a way that per matrix of (160,3) a label is connected of shape (1,6). This merged layer goes through a LSTM layer with 32 hidden units. Finally this goes trough a fully connected layer with 480 output neurons with a Linear activation function and reshape this so that the output of the Generator is of shape (160,3). The loss function of the Generator is a Binary cross entropy and the Adam optimizer with a learning rate of 0.01 with a learning rate decay of 1e-10, β_1 of 0.55 and β_2 of 0.60.

Discriminator
For the Discriminator, there is one input of shape (160,3). This input follows two separate paths in order to get two output layers. In the first path, the input goes through a LSTM layer with 32 hidden units. Followed by a fully connected layer of 1 hidden neuron with a sigmoid as activation function. The output layer is another fully connected layer of 1 hidden neuron with a sigmoid as activation function. In the second path, the input goes through a LSTM layer with 32 hidden units. Followed by a fully connected layer of 6 hidden neurons with a ReLu a activation function. As output layer, another fully connected layer of 6 hidden neurons with softmax as activation function is used. So, the Discriminator has two output layers. Therefore, two loss functions are used. Namely, a Binary cross entropy and a Categorical cross entropy. The SGD optimizer is used with a learning rate of 0.02 and a learning rate decay of 1e-7. The Nesterov accelerated gradient descent is used and a momentum of 0.8.

6.2 Conditional Generative Adversarial Networks Results

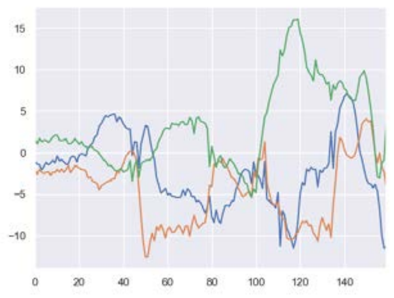
The model versions tested, described in section 6, did not give the desired results. The intention was to create a CGAN that could generate realistic human sensor data of various activities. The problem is that the data generated has often high values and incorrect patterns. When applying more iterations, some peaks with extreme values will occur.

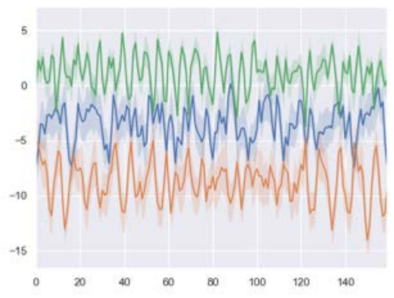
Figure 11 shows the generated data of the CGAN at 600 iterations on the left side and the actual data of the same activity on the right hand side. This data shows the x-, y- and z-accelerometer data in blue, green and red respectively for the activities Jogging, walking the stairs and Walking. These activities are chosen because the Activity Classifier could classify these activities with a fair certainty. When a model is chosen with more iterations, the generated accelerometer values get too high. The actual data is between -20 and 20 and so the generated data should have the same maximum and minimum in order to get realistic human sensor data. Furthermore, the generated data does not look like any real data of any activity described in section 4, Which is also shown in figure 11. The generated data fluctuates too much in comparison to the real data, which is more stable.

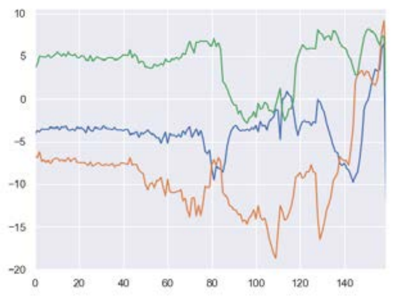
However, figure 12 shows that the sequences generated are just in three cases identified as the correct label by the Activity Classifier. The classifier recognizes the sequence of Jogging with an certainty of 97.5%, the sequence of walking the stairs with an certainty of 56.4% and the sequence of Walking with an certainty of 80.6%. The other three sequences are misclassified completely. So, the more active activities can be generated and classified with the corresponding label, fairly accurate.

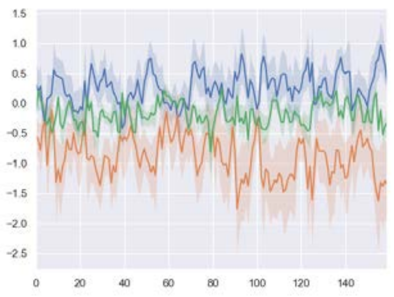
Figure 13a shows the Discriminator- and Adversarial loss of the CGAN for 900 iterations. For the first 350 iterations the losses are relatively stable, then the losses drop. The Discriminator loss drops slightly and the Adversarial loss makes a steeper dive, where the losses becomes stable after roughly 700 iterations. There is a higher variance of the Discriminator loss than the Adversarial loss. The moment at which the losses become stable and the patters of which the loss changes, differs per activity. However, this is a good representation of the behaviour of the losses in general.

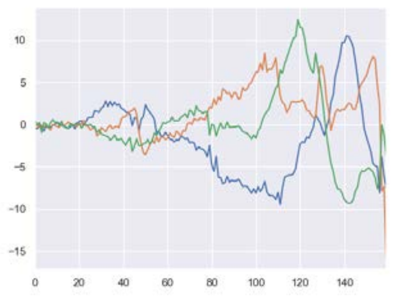
The accuracy of the Discriminator is shown in figure 13b. This figure shows that the Discriminator is correct in approximately 70% of the cases in determining if the data is generated or real. In ideal case the Discriminator thinks all data is real and that will make the accuracy of the Discriminator 50%. Looking at the trend of the accuracy, it indicates that after more iterations, the Discriminator has can identify if the data is generated or real with a higher accuracy. Like mentioned earlier, this is also seen visually. After more iterations, the generated data get too high values as well fluctuates too much. The accuracy of the generated data of all activities are approximately 70% and increasing after more iterations.


(a)


(b)


(c)


(d)


(e)

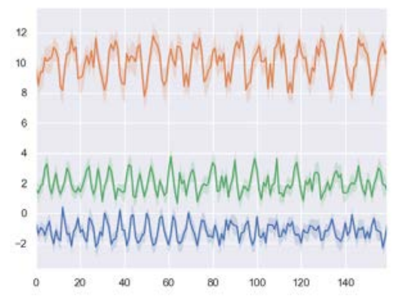

(f)

Fig. 11: Generated (left) and real (right) data of Jogging, Stairs and Walking at iteration 600

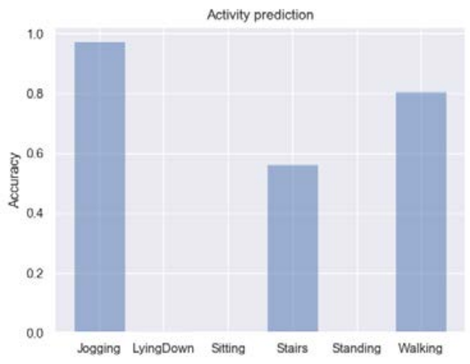
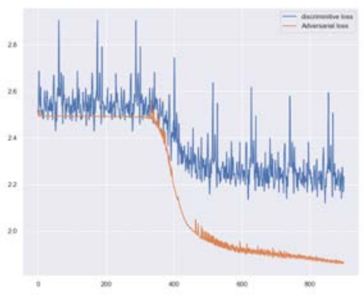
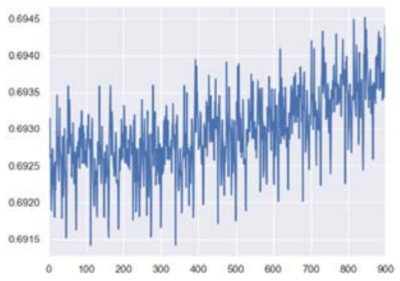


Fig. 12: Correctly classified by Activity Classifier at iteration 600



(a) Loss of Generator and Discriminator



(b) Accuracy Discriminator

Fig. 13: Loss and Accuracy CGAN of Jogging for 900 iterations

In conclusion, it is hard to generate realistic human sensor data. However, it is possible to generate different sensor data per activity. Like described in paper [1], generated data by the Generator of a GAN is performing optimal after a different amount of iterations for each activity. Where the CGAN will generate data for each activity at the same amount of iterations. Which makes it hard to find an optimal number of iterations.

7 Further research

If the intention is there to identify what activity a type of person is performing at a certain moment, it could be interesting to see if there are significant differences per type of person for the accelerometer values. If this is the case, a CCGAN can be created, where an extra condition is added to the CGAN described in previous sections. This condition can describe the type of person. Now data can be generated per type of person per activity.

The data described in section 4 is used to show if there are significant differences between types of individuals. If only the data is included where the participant is known, it results in a dataset of 832,653 datapoints. The number of datapoints per activity are given in table 3. In order to keep enough data

Activity	**Data points**
Jogging	12,995
Lying down	187,695
Sitting	284,589
Stairs	41,825
Standing	105,071
Walking	200,478

Table 3: Number of data points per activity

available to train a CGAN and perform statistical tests, a clustering is applied per category. Each category is divided into three groups. Not every group will be of the same size, however, each group will be large enough to perform statistical tests in order to see if there are differences in the collected data per subgroup. The boundaries of the groups created are given in table 4. The boundaries are chosen randomly so that enough data is available per group. Each individual has an age, height and weight group.

Age group	**Data points**	**Height group (inch)**	**Data points**	**Weight group(lbs)**	**Data points**
(0, 25]	245,379	**(63, 68]**	240,313	**(0, 140]**	288,406
(25, 35]	445,808	**(68, 71]**	434,356	**(140, 210]**	131,773
(35, 55]	135,077	**(71, 77]**	151,595	**(210, 250]**	406,085

Table 4: Number of data points per group

Figure 14 gives the impression that there are differences between the x-accelerometer data per age group for each activity. The boxplots of the x-, y-, and z- accelerometer data of the age, weight and height groups per activity gave comparable results as figure 14.

An analysis of variance (ANOVA) is used to identify if there are significant differences in means between a group within each of the categories age, weight and height. The ANOVA is performed for the x-, y- and z-accelerometer data and for each activity separately.

The result of almost all the ANOVA tests performed is that the p-values of the ANOVA tests are 0 or approximately 0. This indicates that there is at least one group significant different in each category (age, weight and height) for the x-, y-, and z-accelerometer data of each activity separately. The only p value larger than 0.05 is for the category height for the z-accelerometer data of the activity Jogging. This indicates that there is no significant difference between the means of the groups of the category height for the z-accelerometer data of the activity Jogging.

ANOVA only identifies if there is a difference in means of a group of a category, not which group(s) are significantly different to another. The Tukey honestly significant difference (HSD) test is used for further research. The main idea of this test is to evaluate if there are differences between the means of every pairs of groups [15] in a specific category e.g. Age. For example, the HSD test evaluates if the means of the x-accelerometer data differ between the age group (0,25] and (25,35]. This is done for every pair of age group per activity.

Again, almost all the p values calculated are 0 or approximately 0. This indicates that the means of almost every pair of groups in almost all the categories are different from each other. The only p value larger than 0.05 is for the pair of age groups (35, 55] - (0, 25] for the z-accelerometer data for the activity Jogging. This indicates that there is no significant difference between the age groups (35, 55] - (0, 25] for the z-accelerometer data for the activity Jogging.

An explanation that both (ANOVA and HSD test) of the p-values larger than 0.05 are in the activity Jogging could be that there are not a lot of data points for this activity. Data of just five individuals is available, so there are just one or two individuals in a group of a category. This number is too small to look for significant differences between groups.

Overall, the conclusion that individuals divided into groups per category, have different values for their x-, y- and z-accelerometer data per activity, can be made.

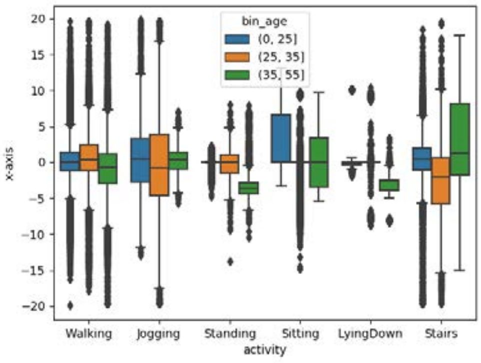


Fig. 14: x-accelerometer data against age group per activity

8 Conclusion

In conclusion, it is clear that the CGAN does not perform well. An explanation could be that while training, every time data of a different activity is used. This makes it harder to look for patterns in the data of a certain activity. Where a GAN, like in [1], should be trained per activity, which makes it easier to find patterns in the data. Another explanation could be, paper [1] shows that the generated data by the Generator of a GAN is performing optimal for each activity after a different amount of iterations. The CGAN will generate data for each activity at the same amount of iterations. However, it is succeeded to create different data per activity with the CGAN. Which is an indication that the CGAN is working, but returns inaccurate data. Expected, because of the research done, is that for this type of data, a CGAN with a structure comparable to the CGAN shown in figure 10b will work optimal.

Because a CGAN was created that could generate different data per activity, in combination with the findings of [1], the expectation is that if the structure mentioned in this paper is followed, a well working CGAN that generates realistic human sensor data could be created. However, more layers should be added and the layers should be enlarged. Furthermore, more research and exploration should be done according to the parameters as well as optimizers, activations and loss functions.

If a well working CGAN is created, then human realistic sensor data of multiple activities, can be generated. This makes it easier and cheaper to get reliable labeled sensor data. Individuals do not have to track their activities while using their smart device to measure their movements, because this data can be generated using the CGAN. this will also exclude any privacy-sensitive problems with the tracked sensor data.

Further research could be to investigate if a "CCGAN" could be created. This is created by adding an extra condition to the CGAN. Where per type of individual, per activity, realistic human sensor data can be generated. However, dividing the data into groups for every type of individual will lead to very small datasets and makes it even more difficult to train. It could be, that this makes the whole training process faster. Since, as shown in section 7, differences per type of individual are significant.

References

1. Ali el Hassouni, Mark Hoogendoorn, and Vesa Muhonen. Using generative adversarial networks to develop a realistic human behavior simulator. 2018.
2. Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. 6 Nov 2014.
3. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. 10 Jun 2014.
4. Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. 2016.
5. Colah. Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/. 27-08-2015.
6. Yuwen Chen*, Kunhua Zhong, Ju Zhang, Qilong Sun, and Xueliang Zhao. Lstm networks for mobile human activity recognition. 2016.
7. Jon Gauthier. Conditional generative adversarial nets for convolutional face generation. 2015.
8. Peng Shen, Xugang Lu, Sheng Li, and Hisashi Kawai. Conditional generative adversarial nets classifier for spoken language identification. 20-08-2017.
9. Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. 2018.
10. Runde Li, Jinshan Pan, Zechao Li, and Jinhui Tang. Single image dehazing via conditional generative adversarial network. 2018.
11. Grigory Antipov, Moez Baccouche, and Jean-Luc Dugelay. Face aging with conditional generative adversarial networks. 2017.
12. Guy Ernest. Combining deep learning networks (gan and siamese) to generate high-quality, life-like images. https://aws.amazon.com/blogs/machine-learning/combining-deep-learning-networks-gan-and-siamese-to-generate-high-quality-life-like-images/. 11-09-2017.
13. Lockhart J.W., Weiss G.M., Xue J.C., Gallagher S.T., Grosner A.B., and Pulickal T.T. Design considerations for the wisdm smart phone-based sensor mining architecture. 2011.
14. G. Chevalier. Lstm human activity recognition. https://github.com/guillaume-chevalier/LSTM-Human-Activity-Recognition. 2017.
15. Herv Abdi and Lynne J. Williams. Tukeys honestly significant difference (hsd) test. 2010.