Master of Aerospace Engineering

Research Project: S2 Project report

Using conditional GANs to develop a realistic human-robot interaction simulator

Author:
Guilherme Penedo

Tutors:
Giorgio Angelotti
Caroline Chanel
Nicolas Drougard

June 2021

Declaration of Authenticity

This assignment is entirely my own work. Quotations from literature are properly indicated with appropriated references in the text. All literature used in this piece of work is indicated in the bibliography placed at the end. I confirm that no sources have been used other than those stated.

I understand that plagiarism (copy without mentioning the reference) is a serious examinations offense that may result in disciplinary action being taken.

Date: 8th of june 2021
Signature:

Abstract

Human-robot systems combine the advantages of humans' adaptability with those of the fast decision making of autonomous agents to improve outcomes for a variety of tasks. Generative models can be used to plan a supervision policy to oversee task allocation in this type of systems, so that this may be done in an optimal manner. The goal of this project is to use one particular type of generative models - Generative Adversarial Networks (GANs) - to create a human-robot interaction simulator capable of generating realistic human-like actions that can then be used to estimate one such policy. A dataset from the "Firefighter Robot Mission" game is used, and a simple GAN model was trained to successfully generate player keypresses with a frequency distribution close to that of the original dataset. This model will be used in future work as a base into which game state's features can be incorporated so that the generated data is dependent (conditioned) on the current game context, using Conditional Generative Adversarial Networks. Other possible avenues for improvement are also discussed.

Keywords: human–robot interaction; mixed-initiative mission; generative models; generative adversarial neural networks; conditional adversarial neural networks

Contents

1 Introduction

To design effective human-robot systems one key approach is to have a dynamic assignment of tasks between the human operator and the automated system. To this end, a policy can be developed to decide when to transfer control from the human to the automated system and the reverse, based on the current state and the operator's mental condition [1].

The system's operation can be modeled as a Markov Decision Process (MDP) to optimize one such policy, as done in [2], using crowdsourced data from an online game where participants have to control a firefighting robot. This yields a probabilistic representation of possible transitions for a given system state [3].

To build upon [2], Reinforcement Learning can be used to create a generative model to mimic a human's behaviour for a particular mission state, thus allowing for the creation of a human-robot interaction simulator that can be used to further optimize a supervision policy, without the logistic difficulties inherent to observing an actual human responding to the different situations.

Since the particular shape of the data corresponding to a human's actions (the data that is to be generated) is not known *a priori*, the recently introduced Generative Adversarial Networks (GANs) [4] will be used on the firefighting robot dataset from [2] to sample from the interaction data's distribution without imposing a prior on its analytical form.

GANs consist of two neural networks: the generator, which takes as input random noise and generates data whose distribution shall ideally approach that of the real data, and a discriminator which classifies data as either real (belonging to the training set) or as generated. The two neural networks have opposing objectives, as the generator seeks to produce data resembling that of the training set, thus rendering the discriminator unable to distinguish between the two, while on the other hand the discriminator tries to improve its ability to predict whether the data comes from the generator or the original dataset [4].

Conditional Generative Adversarial Networks (CGANs) [5] add additional inputs to both the generator and the discriminator, in order to allow the GAN's generated data to be conditioned on some criteria, therefore allowing for the generalization of a GAN without the need to train many different individual GANs. In this project, a CGAN will be used to condition the data generated on certain variables of the firefighting robot's game state, so that the generated data will depend on the current state of the game.

The goal of this project is, therefore, to design and train a CGAN capable of generating realistic human-robot data that may later be used to determine an optimal supervision policy.

The code created for this project can be found on the following github repository:
https://github.com/guipenedo/horizon-ganns.

2 State of the Art

Probability density estimation is the reconstruction of an estimate of the underlying probability density function from which a series of observations were obtained, given a sample of data. Obtaining the probability density function that generated the data can be extremely useful both for the statistical analysis of the phenomenon and for obtaining a useful description of the reality that could be exploited to pursue other purposes: e.g. the optimal control of a learned stochastic decision process [6].

The current approaches that perform probability density estimation can be approximately clustered into explicit and implicit methods [6]. The first tries to explicitly obtain a functional shape for the density of the data, while the second only learn to generate samples that ought to have come from the searched distribution without possessing its analytical expression [7].

One way to perform probability density estimation is through Generative Models. The latter are an implicit method, or hybrid in some cases, that became very popular thanks to the expressivity of Deep Neural Networks [7].

There are currently quite a few different generative models. Fully visible belief networks (FVBN) [8] are one of them: the probability of a vector of data (a sample) is calculated using the probabilities of the vector's individual elements. It is the basis for the WaveNet [9] model which is capable of generating raw audio waveforms, such as natural sounding human speech. Their main drawback is the time taken to generate data: e.g. WaveNet takes 2 minutes to generate 1 second of data [7].

Variational Autoencoders (VAEs) [10] and Normalizing Flows [11] are another kind of generative models. They offer both sampling and density estimation [12], but often result in poor samples being generated when compared with other models [7].

Finally, there is a type of generative models called Generative Adversarial Networks (GANs) which will be the model used in this project. GANs provide greater generation speed than FVBNs and their sample quality, specially when applied to images, is generally well regarded. However, they introduce some drawbacks of their own, such as training instability and mode collapse. A more detailed overview and comparison can be found in [7].

In the following sections different GAN architectures will be discussed.

2.1 Generative Adversarial Networks (GANs)

GANs were first introduced in [4]. They grew in popularity due mainly to their success in generating entirely new images, such as faces of people who do not exist[1].

As briefly mentioned above, a GAN is made up of two different neural networks that will compete against each other: the generator and the discriminator.

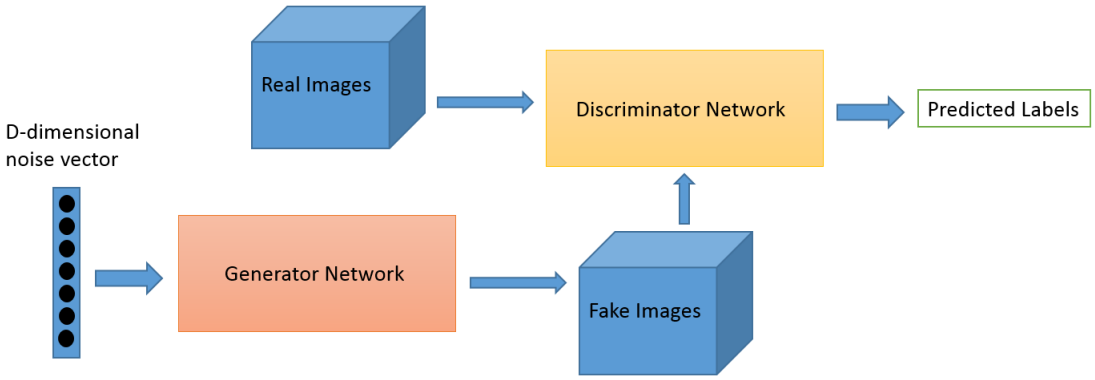

Figure 1: GAN architecture. Credit: Jon Bruner and Adit Deshpande, O'Reilly

A schematic view of this architecture can be seen on Figure 1, in which there is a dataset consisting of images and a GAN is used to generate new realistic images. As such, a generator network taking as input a noise vector and giving as output an image is used along with a discriminator which, given an image, yields as output the probability (in $[0, 1]$) of this image belonging to the "real images" dataset. This is formalized on the loss function used to train the GAN from [4]:

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

In equation (1), p_{data} is the distribution the training data comes from (the real images); x is a single image; z is a latent vector (the noice vector that is the input of the generator) and p_z the distribution it was sampled from (usually a standard normal distribution). $D(x)$ corresponds to the discriminator, and is the estimated probability that sample x comes from p_{data}, while $G(z)$ is an image created by the generator for the input z (which, again, is the noise vector). The entire expression is maximized if the discriminator correctly classifies the real data with a value close to 1 and the generated data with a value close to 0, and therefore the discriminator is trained with this goal. On the other hand, the generator seeks to minimize this expression. In practice, both networks are trained in each training iteration. It should be noted that the generator does not have direct access to the training data and only learns through its interactions with the discriminator [13].

[1] https://thispersondoesnotexist.com/

Ideally, this minmax problem will converge to $p_{data} = p_g$, in which p_g is the distribution of the data generated by the generator, i.e., the GAN will converge so that the generator implicitly learns the distribution of the real data, or, in other words, when the discriminator is unable to tell whether a sample is from the training set or generated. In pratice, this is quite hard to achieve and convergence is still being actively researched.

2.2 Wasserstein GANs (WGANS) and WGAN-GPs

In fact, GAN training is often unstable, and prone to problems such as mode collapse, wherein a large range of values in the latent space (the noise vectors are drawn from this space) are mapped to the same generator's output (basically the variability of the data the generator produces is affected).

Reference [14] introduces a possible avenue to overcome these problems, by using a new loss function, based on a concept known as the Wasserstein or Earth Mover distance, as well as changing some parts of the architecture (the discriminator no longer outputs a probability, for instance). This new GAN is called the Wasserstein-GAN, or WGAN.

The new loss function, based on the Earth Mover distance, is the one in equation 2, where p_g is implicitly defined by $\widetilde{x} = G(z)$, $z \sim p_z(z)$:

$$\min_G \max_D \mathbb{E}_{x \sim p_{data}(x)}\big[D(x)\big] - \mathbb{E}_{\widetilde{x} \sim p_g(z)}\big[D(\widetilde{x})\big] \tag{2}$$

However, for these improvements to be achievable, an extra constraint must be applied to the discriminator (which in this work is called the critic): the Lipschitz constraint. This is quite difficult to optimize and the method chosen by the authors consists in clipping the values obtained.

The WGAN may sometimes still fail to converge or generate poor samples, due in great part to the weight clipping used because of the Lipschitz constraint. To overcome these problems, another work, [15], suggests changes that make weight clipping unnecessary, by introducing a new term in the loss function: gradient penalty. This new model is called WGAN-GP. WGAN-GP serves as a good base to the implementation of GANs.

2.3 Conditional Adversarial Neural Networks (CGANs)

Simple GANs generate data without taking into account any specific context or state information. One such example is the generation of black and white pictures representing handwritten images (based on the MNIST dataset[2]): a GAN trained with this dataset will generate new images resembling the ones it was trained with but, however, there is no way to request the GAN to only generate images of a particular digit, without training 10

[2]https://en.wikipedia.org/wiki/MNIST_database

different GANs, one for each digit.

Conditional Adversarial Neural Networks (CGANs), were introduced in [5] and extend traditional GANs by appending to the generator and discriminator's inputs the data on which the GAN is to be conditioned.

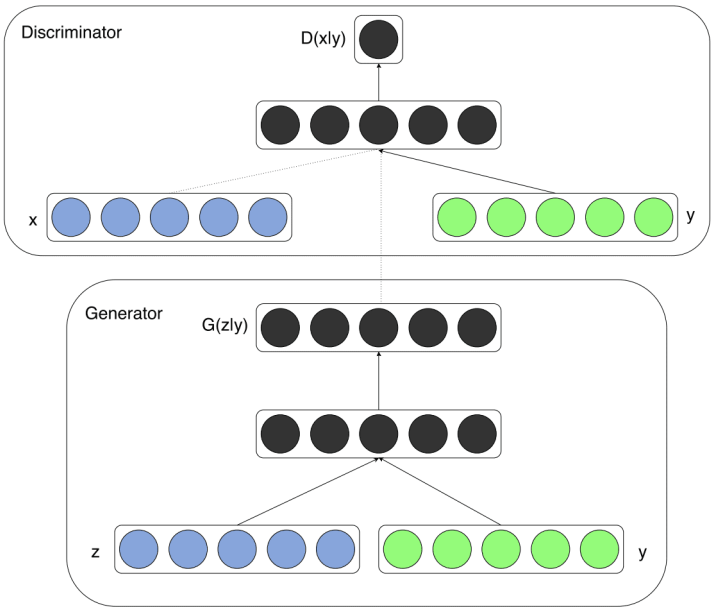

Figure 2: CGAN architecture. Credit: [5]

In Figure 2, z is, as before, a latent vector (a vector containing noise/random values), and x is a vector containing a sample's data. However, a new vector, y is introduced, which is the condition. The condition is thus incorporated into both the discriminator and the generator, which discriminate and generate, respectively, under this condition. For example, for a CGAN trained with the MNIST dataset, $D(x|y)$ would not be simply the estimated probability that image x belongs to the training dataset, but the estimated probability that image x belongs to the training dataset given that it is meant to represent digit y.

Equation (3) reflects these changes to the original loss function of equation (1).

$$\min_G \max_D \mathbb{E}_{x\sim p_{data}(x)}\big[logD(x|y)\big] + \mathbb{E}_{z\sim p_z(z)}\big[log(1-D(G(z|y)))\big] \tag{3}$$

This principle, and CGANs in general, can be adapted to have a single model generate data for different conditions or states, for instante, for the automatic or manual control modes of a human-robot system.

3 Key issues and novelty

Generating human data, which by nature is full of uncertainty, multi-modal and whose distribution may have an arbitrary shape, is a big challenge.

GANs are a recent technology (first introduced in "Generative Adversarial Networks" [4], in 2014). They allow us to infer a distribution without imposing a prior on its analytical form and are therefore a good fit for the problem at hand.

GANs have become very popular in image based applications; however, the literature is still scarce concerning applications to other contexts. One of the main issues of using GANs outside of image based applications is that validation becomes much more difficult: it is no longer possible, for the most part, to simply look at the generated image of the network to assess its quality.

Some notable exceptions are "Generating Synthetic ECGs Using GANs for Anonymizing Healthcare Data" [16] (from this year) and "PGANs: Personalized Generative Adversarial Networks for ECG Synthesis to Improve Patient-Specific Deep ECG Classification" [17]. This last reference uses a custom loss function, optimized specifically to reward the resemblance of the generated data's shape to that of the characteristic ECG (eletrocardiography)'s shape. The creation of new custom made loss functions is another possible avenue associated with this research, which contributes to the project's novelty.

Convergence of GANs is another one of the key issues. As demonstrated above, it is quite difficult to stabilize the network during training.

Regarding the training data, one big issue is its representation and treatment. Human related data is complex and multi-modal. While simplifications likely increase the models' success prospects, they result in the loss of information. A careful balance has to be found in order not to lose too much representation capacity while not making the problem intractable.

4 Firefighter Robot Mission (FRM)

One of the main challenges of the implementation of mixed human and robot (artificial agents) is the supervision policy that dictates task allocation.

An experiment was developed in [1], in which a game, called Firefighter Robot Mission[3], was set up in order to collect behavioral and psycho-physiological features from human operators.

The game itself has a graphical user interface (GUI), pictured in Figure 3, through which the player is expected to control the movement of a robot that has a water deposit, in order to extinguish fires on nearby trees.

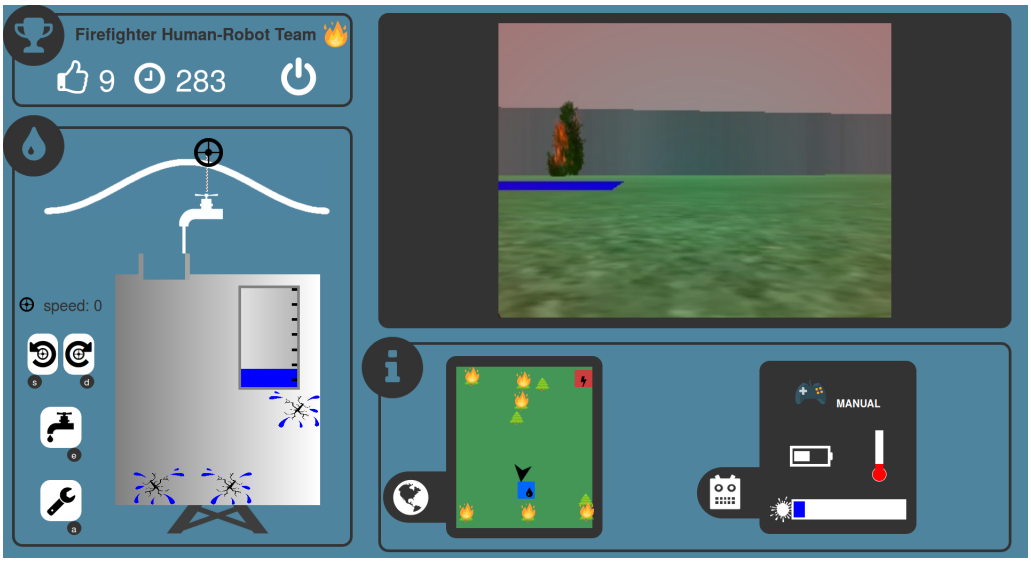

Figure 3: Game GUI. Credit: [1]

On the top right, a simulated viewpoint from the robot can be seen, while on the bottom right there are two panels: one minimap, showing the state of the 9 trees, the robot position and the places where the robot can refill its water tank (the blue square) and recharge its battery (the right square) and another panel showing the current control mode (manual or automatic, the robot's battery, temperature and water tank level). On the left, we can see the ground tank which corresponds to the blue square on the minimap, where randomly leaks appear, which the player must fix, and where the tap that refills this ground tank has to be activated while dealing with an unstable equilibrium point.

The robot's movement is controlled through the 4 arrow keys, while the space key is used to shoot water, with which the robot can extinguish the fires. The mouse is meant to be

[3]http://robot-isae.isae.fr

used to fix the leaks and to refill the water tank (by interacting with the left panel).

A dataset containing game state (which trees are on fire, battery, temperature, water levels, etc) and player actions (keys pressed and clicks made) is available online[4]. One of the end goals of this research project is to be able to apply GAN models to data from this dataset, in order to better estimate an efficient supervision policy.

The structure of the data in this dataset is thoroughly detailed in [2]. Each game played is stored in a csv file. Each line of the csv file corresponds to a 1 second interval of the game. The data in each line contains both the game state and the human's response.

4.1 State data

The state consists on the following: the *remaining_time*, which starts at 600 (each game has a max duration of 600 seconds); *robot_mode*, one of the most important values given the context of the experiment, which is either 0 (manual) or 1 (automatic); the robot position and heading, made up of *robot_x*, *robot_y* and *robot_theta*, with values in the ranges $[-20, 20]$, $[-20, 20]$ and $]-\pi, \pi]$ respectively; the *forest_state*, a binary string (bitmask) with a 1 for trees on fire and 0 otherwise; *battery_level*, $\in [0, 100]$; *robot_temperature*, $\in [20, 240]$, *water_robot_tank* and *water_ground_tank*, both in $[0, 100]$ and *leaks_state*, which is a binary string similar to *forest_state*, detailing the state of the water deposit's leaks.

4.2 Human interaction data

The data saved pertaining to the human's interaction mostly comes down to two columns: *keys*, which is a sequence of keypresses from the set $\{"front", "back", "left", "right", "space"\}$ and *clicks*, a sequence of locations clicked from the set

$$\{"left", "right", "push", "wrench", "leak_i"_{i=1,\dots,9}, "rm\_alarm"\}$$

4.3 Existing preprocessing

As can be seen, there's quite a lot of data and one of the main difficulties it will pose will be on the way in which it is to be encoded/normalized. In [2], the possible (x, y) positions were divided in 9 (3×3) sectors: $\{C, N, E, S, W, NE, NW, SE, SW\}$; battery and water levels were simplified to two possible qualitative labels: "nominal" and "low"; the forest_state was used to obtain a simple integer representing the number of fires, as well as some other state variables were chosen, such as a value representing the human's intention (in terms of target position). All in all, with all the variables considered, the

[4] https://personnel.isae-supaero.fr/isae_ressources/caroline-chanel/horizon/

state space has a size of 12960. This preprocessing was not used in our own work.

4.4 Previous work

Another previous research project addressed this same subject. In "Improving Human-Robot Interactions with Deep Reinforcement Learning" [18], the authors sought to leverage the power of convolutional neural networks (CNNs), a widely adopted supervised learning technology [19], on its recent unsupervised learning form - Deep Convolutional Generative Adversarial Networks (DCGANs) [20] - which has had considerable success on image based applications, to train a generator on the FRM dataset.[5]

One of the great advantages of using DCGANs to generate images is that, when it comes to validation, simple visual observation of the generated images is enough to assess whether the generator is being trained properly and its output is improving. Applying DCGANs to non-image data, and in particular to data from the Firefighter Robot Game dataset, presents a number of validation issues, which are also present on the other different GAN architectures.

4.4.1 Networks used

In this project, a DCGAN was implemented to generate robot trajectories, that is, sequences of (x, y, θ) tuples. Trajectories were chosen as a way to somewhat simplify this very complex problem.

After some preprocessing, which focused mainly on normalizing the data, by making the continuous variables be in the range $[-1, 1]$, different approaches and architectures were tested.

For the initial approach, the network was fed the game state (robot position and angle, tree states, tank levels, temperature and battery level), corresponding to 16 different values, and the output was the actual robot displacement and the angle of this displacement.

There was a second architecture tested, and finally a third one, which was quite similar to the second one but with a few changes. This last architecture generated the aforementioned (x, y, θ) tuples, by analyzing and generating chunks of 10 seconds worth of gameplay data. This network was conditioned on a single value: the *robot_ mode* (autonomous or manual).

[5] Their work is available at https://github.com/horizon-dcas14/project

4.4.2 Results

These sequences of 10 seconds worth of robot positions were then plotted and, even though there were some inconsistencies (the direction the robot was facing was not consistent with its movement, for instance), it could be concluded that the network was actually able to generate some realistic looking data.

The focus on trajectories offers one very clear advantage: as the robot's movement when under autonomous mode is dictated by an algorithm (go to the nearest tree on fire, refill tank if the water level is below a certain threshold, etc), the results could be validated with this algorithm. Then, there would be a greater confidence in the human data generated from the same network.

4.4.3 Limitations

In terms of limitations, using only the robot_mode (thus a single input) for the condition does not allow the model to learn the complex game dynamics associated with the actual task of putting out fires and how an operator moves the robot towards the trees or the water/battery charge locations.

Furthermore, trajectories do not directly represent a human's actions, but instead are a consequence of them (for example, pressing the front key multiple times will result in a certain displacement). For the automatic mode, this is not even true: a human operator can press multiple keys and only the automatic algorithm will dictate how the robot will move. However, the data associated with a human's actions during automatic mode can still have value, as it could allow an assessment of the human's mental state or whether he even realized that the control mode changed in the first place.

Finally, if the end goal is to create a realistic human-robot interaction simulator, once more game state related variables are incorporated into the model, the choice to consider 10 second intervals becomes problematic, as the state of each row depends on the actions taken on the previous row, and therefore it would not be possible to know the game state for the full 10 seconds *a priori*. This is, however, likely the hardest limitation to overcome, as it simply is quite difficult to train a model without incorporating previous data or a chunk of data, due to the time dependency the problem displays (multiple seconds worth of data allow us to properly analyse the human's intentions, for instance).

5 Project timeline

5.1 Initial phase

For this project, "Using conditional GANs to develop a realistic human-robot interaction simulator", on an initial phase, the author sought to learn and familiarize himself with the realm of machine learning, starting by taking a highly rated course in Coursera [21], from Stanford University.

Then, the library that will be used in the project was chosen: PyTorch[6] - one of the most popular python libraries for machine learning. Becoming used to this library involved following some online tutorials in order to fully grasp how it is structured and may be used.

5.2 Literature review

The next phase, which took a considerable amount of time, was the literature review.

Starting with the original "Generative Adversarial Networks" by Ian J. Goodfellow [4], then continuing with "Conditional Generative Adversarial Nets", by Mehdi Mirza and Simon Osindero[5], "Wasserstein GAN" by Martin Arjovsky *et al.* [14] and "Improved Training of Wasserstein GANs" by Ishaan Gulrajani, Faruk Ahmed, *et al.* [15].

Since most of these works are quite theoretical and pertain to a domain that was unknown to the author, fully understanding them was a time-consuming process.

Some other literature was consulted, mainly "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks" by Alec Radford *et al.*[20].

During this phase, the MNIST dataset, which contains black and white pictures of handwritten digits on a small scale, was used to test the concepts learned regarding GANs and CGANs. In the first case, a GAN was trained to generate new handwritten digits, while in the second one this same network was modified to include 10 additional inputs (the condition), where the desired digit was encoded in a one-hot configuration (for example, to obtain a 0 this input would be $[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]$). Both implementations are available on a *github repository*[7] and sample data from these implementations can be seen on Figures 4 and 5 below.

[6]https://pytorch.org/

[7]https://github.com/guipenedo/horizon-ganns/tree/main/mnist

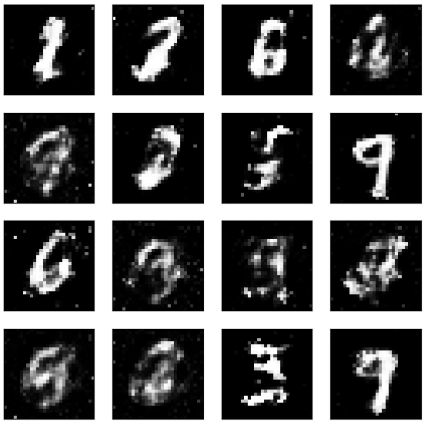

Figure 4: Sample data generated by the GAN

On the simple GAN architecture from which the images in Figure 4 were generated, it is not possible to generate images of a specific digit.

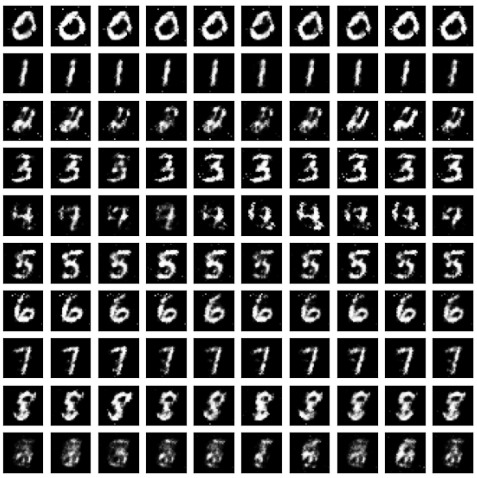

Figure 5: Sample data generated by the CGAN, each row conditioned on the same digit

This is, however, possible with a CGAN architecture, such as the one in Figure 5, where the condition given to the generator to obtain these images was 0 for the first row, 1 for the second, 2 for the third, etc. In this image we can also see for some digits a phenomenon known as mode collapse, which is when the range of images produced is quite small due to the success of one particular image in fooling the discriminator.

5.3 Firefighter Robot Mission

After obtaining an understanding of the underlying concepts and implementations of GANs, the next step was to start looking into the FRM dataset.

In terms of what data should be generated, the choice differed from the one in [18], which was trajectories (tuples of (x, y, θ)). For this project, the idea is that the models used would generate keypresses, which are what actually dictates the robot's positions. Ideally, the actual sequences of keypresses would be generated, but since these are variable in length, and can in some cases be quite large (as many as 40 consecutive keypresses in a single second, probably corresponding to the player holding down on the key instead of just pressing it) a simplification was implemented: instead of generating the sequence of keys, the model would generate the number of times each of the keys was pressed.

It was decided that this model should consider 1 second intervals (corresponding to one line of the csv files in the dataset), instead of a chunk of multiple second intervals. As such, Convolutional Neural Networks (CNNs) were not used. However, in an effort to introduce some information regarding the previous game state, so that player intentions could be taken into account and some sense of continuity could be considered, 3 new columns were introduced: the difference/displacement from time interval $t-1$ to time interval t in the robot's position and heading, *robot_x_diff*, *robot_y_diff* and *robot_theta_diff*.

There are 5 keys: "front", "back", "left", "right" and "space". Therefore, the network will have 5 outputs. As the Tanh activation function was chosen for the last layer of the generator, the data corresponding to these 5 keypresses had to be normalized in the $[-1, 1]$ range. The highest number of times any key was pressed in a single second was 37, and so the number 40 was chosen to be mapped to 1 while 0 (the minimum for any key) would be mapped to -1.

Regarding the game state variables to be considered, which would then be the condition to our CGAN, initially, the following variables were selected: "remaining_time", "robot_mode", "alarm", the robot position ("robot_x", "robot_y", "robot_theta", "robot_x_diff", "robot_y_diff" and "robot_theta_diff"), the forest state (which was converted into 9 individual binary values), "battery_level", "temperature" and "water_robot_tank". These values were also normalized so that they would be centered around 0 and have values between -1 and 1. This makes up quite a large condition vector, with a total of 21 variables.

To be able to properly visualize the data, a small GUI representation was created. It displays the information contained in this selection of state variables, as well as the number of keypresses. An example for a sample from the dataset can be seen in Figure 6. In this sample, there are 287 seconds remaining in the game, we can see the robot's battery and water level (67% and 20%, respectively), the game is currently in manual mode (thus being controlled solely by the player) and the "front" key was pressed twice. We can also see that there are currently 6 trees on fire (the trees are the green triangles, if they have a smaller

orange triangle inside them, then they are on fire), the robot (black triangle) is moving towards the bottom right trees and we can also see the locations where the water is refilled (blue square) and the battery is recharged (top right, in red).

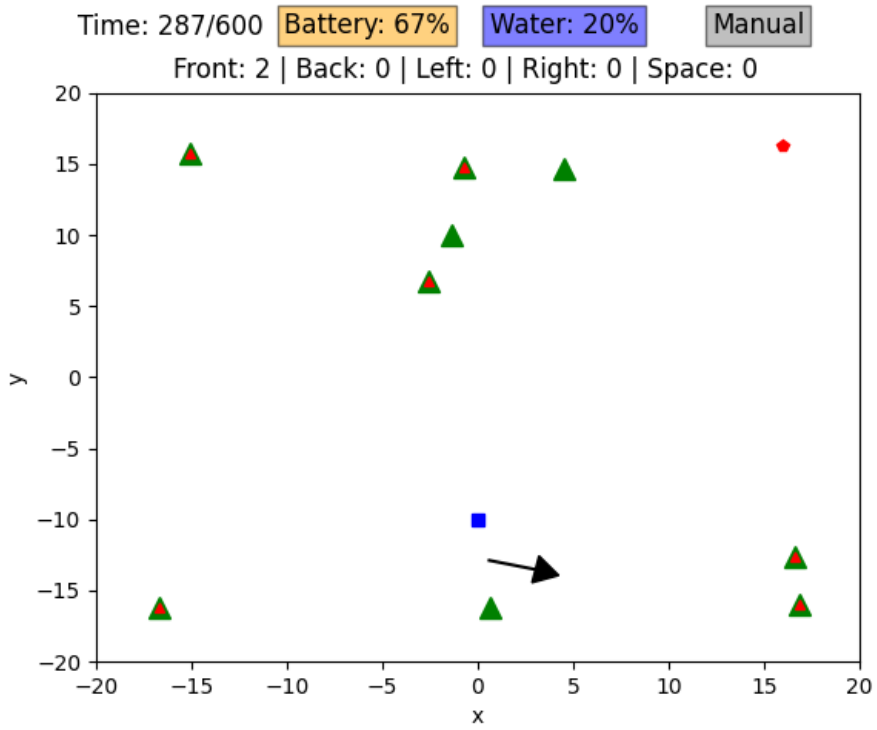

Figure 6: A sample from the dataset displayed on the GUI

5.3.1 First attempt at a CGAN

A first model was created: a CGAN architecture using the standard binary cross entropy loss function (equation 3). In the hidden layers of the generator and discriminator the activation function used was the "LeakyReLU" function (Leaky version of a Rectified Linear Unit); as already mentioned no convolutional layers were used, and so the two networks were based on dense/linear/fully-connected layers. Dropout layers were also used to improve training.

This first model did not produce good results: the data generated was either 0 keypresses or a very high value (near the maximum, or even the maximum, of 40). Most of the times the data generated consisted uniquely of 0s. This could possibly have been caused by the fact that on most samples, by a large margin, there are a total of 0 keys pressed. To investigate this possibility, the network was trained using only a subset of the original dataset: only the samples where at least one key was pressed were used for training.

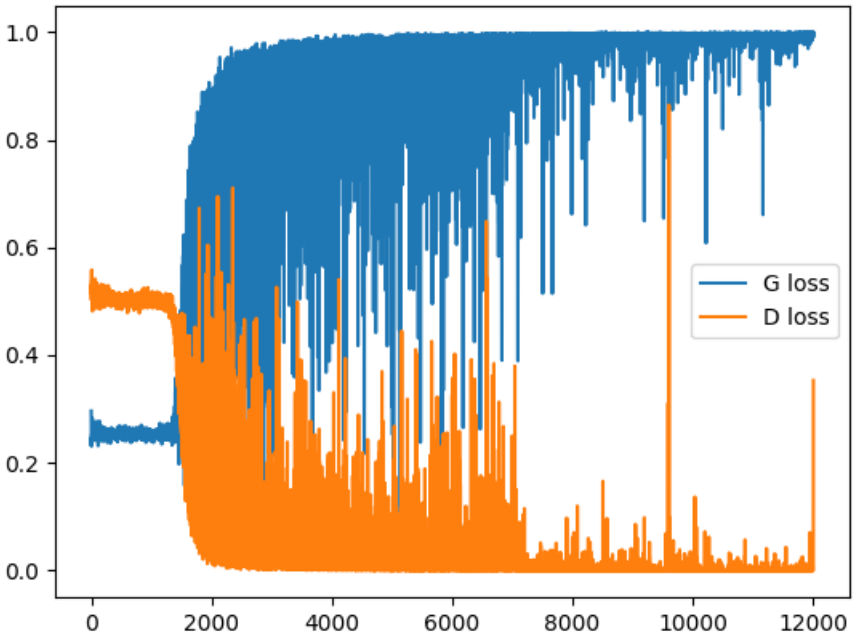

Figure 7: Generator and discriminator losses during training of the first model.

As can be seen in Figure 7, the discriminator loss went to 0 while the generator's loss approached 1. A discriminator loss of 0 means that the discriminator can discriminate between the real and generated samples without any issues, which is obviously not what is intended. What was happening in this particular situation was that the generator continued to just generate 0 keypresses, while the training dataset consisted of only samples where there was at least one keypress. This then made it quite easy for the discriminator to fulfill its function. This issue, where the discriminator's loss approaches 0, is quite common in GANs, which are in general hard to train, and is known as convergence failure[22].

Different loss functions were tried, such as the mean squared error (MSE), as well as some small changes to the architecture, but the results were, unfortunately, similar.

5.3.2 Data analysis and simple GAN

After the disappointing results of the first attempts, another look was taken at the dataset to try to gather some helpful insights.

After analysing the data in the dataset, it could be noted that about 20% of games did not include a single keypress. There are various possible reasons for this, from a game being

used for demonstration purposes to the players not fully understanding how the game works in the first place, but they do not provide an accurate representation of a normal player's gameplay. As such, all the games where no key was pressed, in any time during the entire game, were no longer considered for training. From the remaining, the frequency distribution of the number of keypresses per key was studied. In Figure 8, the frequencies of the "front" key (used as an example) can be seen.

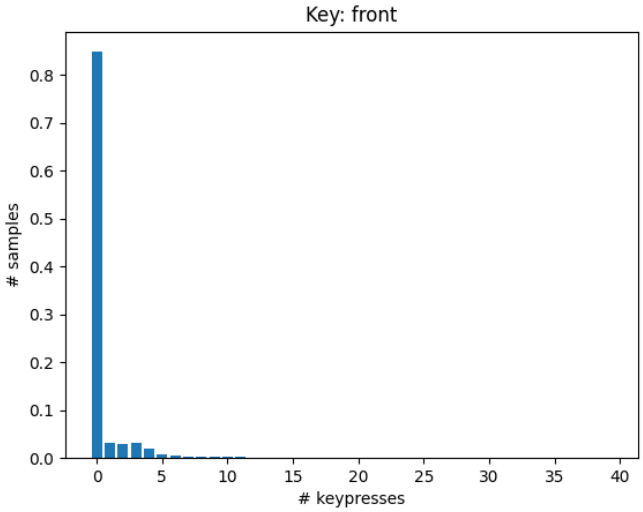

Figure 8: Frequency analysis of the "front" key in the original dataset

We can see in Figure 8 that the majority of samples do not include the "front" key at all (around 85% of all samples), while we have a decreasing count of samples as the number of keypresses per second increase (as was to be expected). We can also conclude that there is only a small number of samples where the "front" key is pressed more than 10 times in a single second, and these may be attributed to latency differences. There is not any clear benefit in considering values higher than 10, whose physical meaning is also quite unclear, and so as a way to simplify the data, samples with more than 10 keypresses of a certain key had that value capped to 10. This is one first improvement compared to the assumptions of the previous model, where we were considering the possibility of having 40 keypresses. The resulting frequency distributions of the "front", "back", "right" and "left" keys in the dataset after this pre-processing were plotted in histograms in Figure 9. It should be noted that the small bump on "10" is due to the grouping of all values greater than 10 on that value.

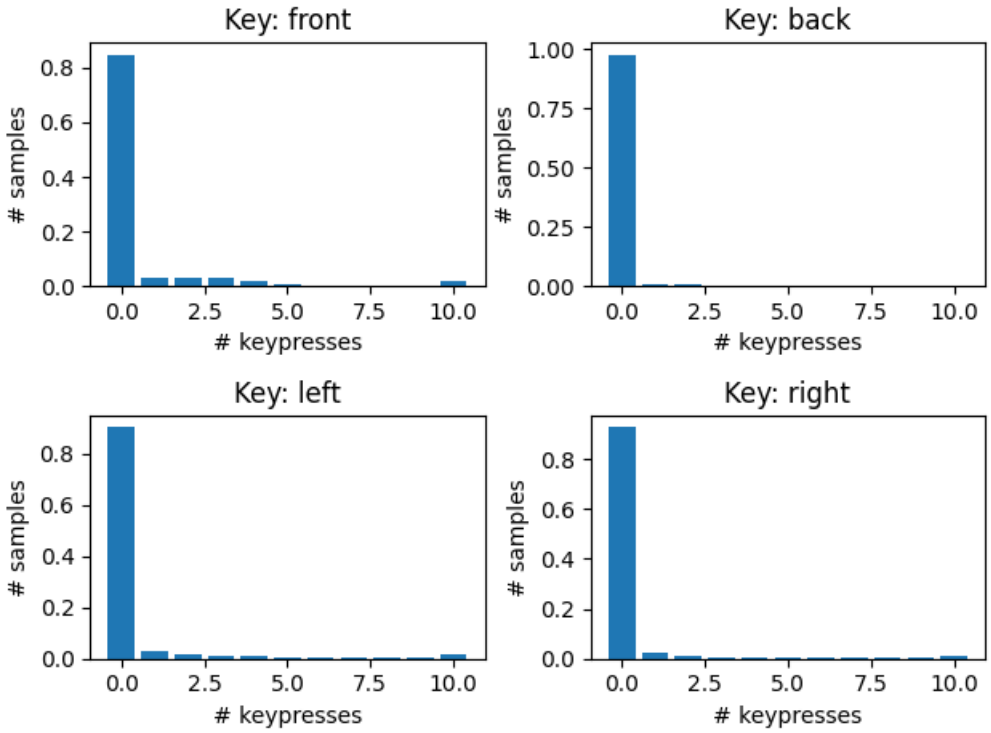

Figure 9: Frequency analysis of the keys in the original dataset (after pre-processing)

In order to carefully verify the behaviour of the network, a simple GAN (without condition) was created to generate keypresses. The goal was to, after training, generate a large number of samples and compare the keypress count frequency distributions from the generated data with that of the original dataset.

Some relevant changes were made. Instead of having the output of the generator be 4 numbers (the "space key" was not considered as dataset analysis showed that it was never pressed more than 1 time), in the $[-1, 1]$ range, 4 vectors of 11 elements, one vector for each key, would be the new output. Instead of the Tanh activation function, the new activation function became the Softmax function, applied along each vector, which normalizes them in such a way that the total sum of the 11 values in each vector would equal 1. In practice, for each key, the network would generate 11 numbers: where the i-th number is the probability that, for this sample, the pressed key is "i". We can then simply take the "i" for which the probability is highest and choose that number as the actual output.

This poses a few problems. Firstly, argmax (the function which consists in choosing the "i" with the highest probability associated to it) is obviously not differentiable, and therefore it is not possible to apply backpropagation in this situation, rendering the training of the model impossible. The only possibility seems to be to keep the 4×11 values during training, and feed them as input to the discriminator (instead of just 4 with the actual values)

but, since the real data is also fed to the discriminator as input, there is a new problem: the actual real data does not consist on probabilities for the count of each keypress, but rather on the actual number of keypresses. Simply turning this information into a one hot vector could bias the discriminator and make it artificially easier to distinguish between real and generated data. To avoid this problem, noise sampled from an exponential distribution with a rate of 9999 was added to the one-hot vector of the real data, which was then turned into a probability by dividing each element by the total sum of all elements.

Initially, a "vanilla" GAN architecture was used. However, later on gradient penalty (from WGAN_GP) was implemented and the loss function was changed (to the one also from WGAN_GP, instead of binary cross entropy). The generator was also only trained every 5 iterations while the discriminator was trained on all iterations. Figure 10 shows the evolution of the discriminator and generator's losses during training.

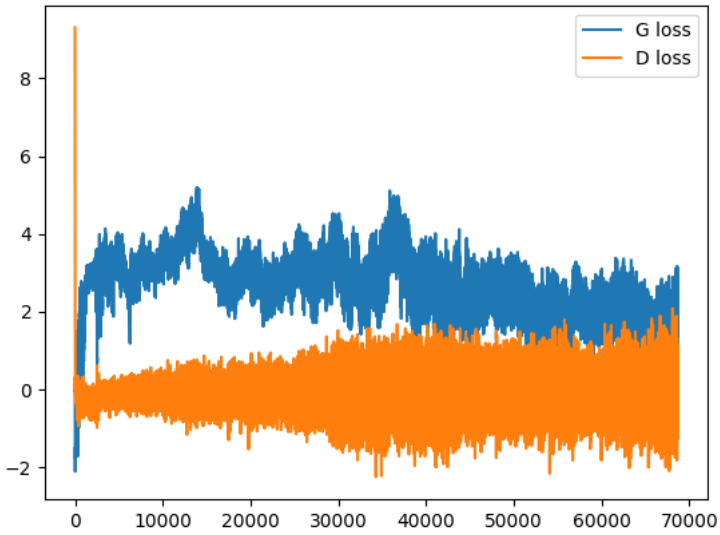

Figure 10: Generator and discriminator losses during training of the WGAN_GP model without condition

After training, hundreds of thousands of samples were generated. For comparison with Figure 9, the resulting frequencies of the keys in the generated data can be seen on Figure 11.

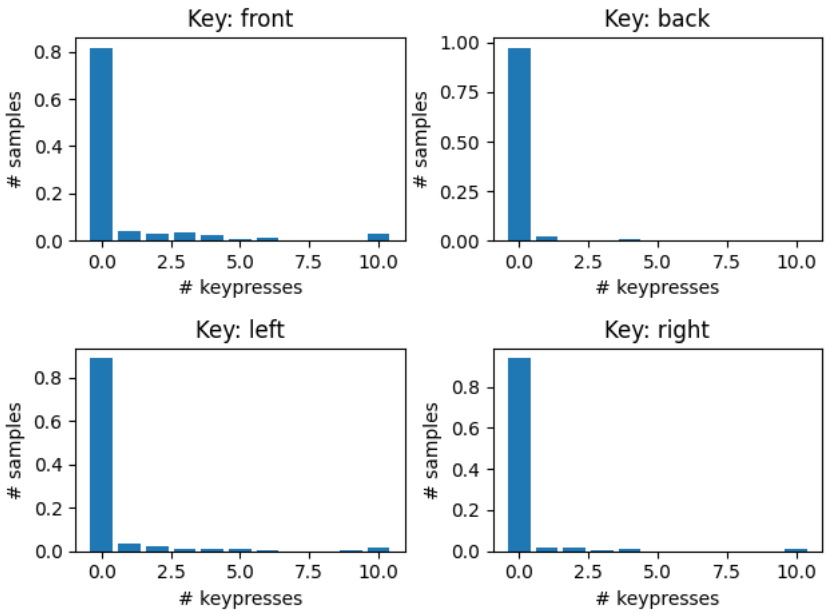

Figure 11: Frequency analysis of the keys in the generated data

As can be seen, the frequency distributions of the generated data closely resemble thos se of the original dataset. From this analysis we conclude that the GAN managed to learn the underlying distribution of the dataset to a certain degree.

To compare the initial, "vanilla" GAN with the modified model following a WGAN_GP architecture, the frequency distributions for each key generated by the two models were compared to those of the original dataset. Table 1 shows the Wasserstein distance[8] between the frequency distributions of each model for a given key and that key's frequency distribution in the original dataset. It can clearly be seen that the WGAN_DP produces samples with a frequency distribution closer to that of the original dataset, as the distance between these two distributions is significant lower than with the vanilla GAN.

Model used to generate samples	Key to compute Wasserstein distance			
	"front"	"back"	"left"	"right"
Vanilla GAN	0.04833	0.03722	0.05902	0.10258
WGAN_GP	0.00674	0.00207	0.00371	0.00299

Table 1: Wasserstein distance for two types of GANs

These are overall good results for the samples generated from the WGAN_GP model. It constitutes a promising base model that may be used towards slowly adding conditional modes to the network.

[8]https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wasserstein_distance.html

6 Conclusion and Future work

The base WGAN_GP model mentioned in section 5.3.2 can be slowly expanded to start including conditions based on the game state, such as the control mode (in theory, the number of keyspresses in autonomous mode should be way smaller than the equivalent in manual mode), the trees state, water and battery level, etc. One of the main issues concerns the robot position, which is hard to incorporate into the model.

The preprocessing and representation in the network of the data from the Firefighter Robot Mission is in itself a key issue. For instance, representing the actual sequences of keys pressed during the game would offer important contextual information (the ordering in and of itself is relevant) but pose significant problems. The same can be said for the different values making up the game's state variables: using a discretization such as the one used in Section 4.3 would likely simplify the problem but result in reduced accuracy due to the loss of information.

One possible approach to improve continuity and to get the model to properly consider previous timestamps is exposed in "Generating realistic human sensor data using Conditional Generative Adversarial Networks"[23], which makes use of Recurrent Neural Networks (RNNs) - a type of network which works in a sort of loop, where output for one timestamp is reused as input to the following timestamp - and in particular Long Short Term Memory (LSTM) networks. Using one such approach contextual results have the possibility to improve, therefore making the simulator behave in a more coherent way.

To improve convergence of GANs, optimizing architecture parameters and hyperparameters (such as learning rate) is crucial. Further study and experimentation to optimize these parameters might improve the results [24].

Another possible avenue to be explored is the use of Convolutional Neural Networks (CNNs)[19], which, as previously mentioned, have shown good results in an image context. Adding some pre-processing that allows the data to resemble an image in shape might make this approach viable.

There is another important part to the simulator that as of yet has not been too explored: the clicks on the leftmost panel of the game's GUI, to control the water refillment of the robot's water tank. Once satisfactory results have been obtained from the keypresses simulator, the architecture may be replicated and slightly adapted to the "clicks" application.

Finally, coming up with additional metrics for data validation is another key goal, in order to surpass the difficulties inherent to data validation on this type of application of GANs.

References

[1] Caroline Ponzoni Carvalho Chanel, Raphaëlle N. Roy, Frédéric Dehais, and Nicolas Drougard. "Towards Mixed-Initiative Human-Robot Interaction: Assessment of Discriminative Physiological and Behavioral Features for Performance Prediction". In: *Sensors, Special issue Human-Machine Interaction and Sensors* 20.1 (Jan. 2020), pp. 1–20. DOI: 10.3390/s20010296. URL: https://oatao.univ-toulouse.fr/25267/.

[2] Jack-Antoine Charles, Caroline P. C. Chanel, Corentin Chauffaut, Pascal Chauvin, and Nicolas Drougard. "Human-Agent Interaction Model Learning Based on Crowdsourcing". In: *Proceedings of the 6th International Conference on Human-Agent Interaction*. HAI '18. Southampton, United Kingdom: Association for Computing Machinery, 2018, pp. 20–28. ISBN: 9781450359535. DOI: 10.1145/3284432.3284471. URL: https://doi.org/10.1145/3284432.3284471.

[3] Mausam and Andrey Kolobov. "Planning with Markov Decision Processes: An AI Perspective". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6.1 (2012), pp. 1–210. DOI: 10.2200/S00426ED1V01Y201206AIM017. eprint: https://doi.org/10.2200/S00426ED1V01Y201206AIM017. URL: https://doi.org/10.2200/S00426ED1V01Y201206AIM017.

[4] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[5] Mehdi Mirza and Simon Osindero. *Conditional Generative Adversarial Nets*. 2014. arXiv: 1411.1784 [cs.LG].

[6] Shakir Mohamed and Balaji Lakshminarayanan. "Learning in implicit generative models". In: *arXiv preprint arXiv:1610.03483* (2016).

[7] Ian Goodfellow. "Nips 2016 tutorial: Generative adversarial networks". In: *arXiv preprint arXiv:1701.00160* (2016).

[8] Brendan J. Frey, Geoffrey E. Hinton, and Peter Dayan. "Does the Wake-Sleep Algorithm Produce Good Density Estimators?" In: *Proceedings of the 8th International Conference on Neural Information Processing Systems*. NIPS'95. Denver, Colorado: MIT Press, 1995, pp. 661–667.

[9] Aaron van den Oord et al. "Wavenet: A generative model for raw audio". In: *arXiv preprint arXiv:1609.03499* (2016).

[10] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. "Stochastic backpropagation and approximate inference in deep generative models". In: *International conference on machine learning*. PMLR. 2014, pp. 1278–1286.

[11] Danilo Rezende and Shakir Mohamed. "Variational inference with normalizing flows". In: *International Conference on Machine Learning*. PMLR. 2015, pp. 1530–1538.

[12] Ivan Kobyzev, Simon Prince, and Marcus Brubaker. "Normalizing flows: An introduction and review of current methods". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020).

[13] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. "Generative adversarial networks: An overview". In: *IEEE Signal Processing Magazine* 35.1 (2018), pp. 53–65.

[14] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. arXiv: 1701.07875 [stat.ML].

[15] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. *Improved Training of Wasserstein GANs*. 2017. arXiv: 1704.00028 [cs.LG].

[16] Esteban Piacentino, Alvaro Guarner, and Cecilio Angulo. "Generating Synthetic ECGs Using GANs for Anonymizing Healthcare Data". In: *Electronics* 10.4 (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10040389. URL: https://www.mdpi.com/2079-9292/10/4/389.

[17] Tomer Golany and Kira Radinsky. "PGANs: Personalized Generative Adversarial Networks for ECG Synthesis to Improve Patient-Specific Deep ECG Classification". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (July 2019), pp. 557–564. DOI: 10.1609/aaai.v33i01.3301557. URL: https://ojs.aaai.org/index.php/AAAI/article/view/3830.

[18] Felipe Teixeira da Silva, Alöıs Blarre, Caroline Chanel, and Nicolas Drougard. *Improving Human-Robot Interactions with Deep Reinforcement Learning*. 2019.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Commun. ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[20] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG].

[21] Andrew Ng. *[MOOC] Machine Learning*. Coursera. URL: https://www.coursera.org/learn/machine-learning.

[22] Jason Brownlee. *How to Identify and Diagnose GAN Failure Modes*. Jan. 2021. URL: https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/.

[23] Maarten Kleinrensink, M. Hoogendoorn, and Ali el Hassouni. "Generating realistic human sensor data using Conditional Generative Adversarial Networks". In: 2019.

[24] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).