POPULATION SYNTHESIS WITH GENERATIVE ADVERSARIAL NETWORKS

Germans Savcisens (s170279) || ***Supervisor:*** *Sergio Garrido (shgm)*

DTU Compute

ABSTRACT

Population synthesis is a large research topic that is focused on generating data that can realistically represent population. Recent research within the Deep Learning, specifically, Wasserstein GAN (WGAN) was able to outperform more common techniques for the population synthesis providing a high fidelity and much more diverse samples. Current paper investigated alternatives to the WGAN that can further increase the quality of the synthesised data. During the project we [1] implemented and evaluated WGAN with Gradient Penalty and Multi-Agent Diverse GANs; on top we added the Gumbel Softmax activation. The results showed that WGAN-GP might be a better solution in comparison to WGAN, as it is able to fit to the high dimensional distribution of data, as well as more stable training.

Index Terms— Population Synthesis, neural networks, generative model, generative adversarial networks

1. INTRODUCTION

Population Synthesis strives to generate synthetic but realistic samples of a population. This also includes the generation of the *micro-agents* [1], i.e. samples representing an individual. Thus, the goal is to provide the ability to sample from a distribution that has similar statistical properties [2] as if it is sampled from the real one.

Due to the privacy issues and data collection techniques (e.g. usually collecting data through surveys), the whole data acquisition process becomes quite complex [3]. While the data might include a big variety of features (e.g. household state, income data, possession of different assets etc.), it exists in a limited amount [3]. The topic is of the interest as many areas relies on the agent-based models; such areas might be part of the urban planning, transportation, geography [1] etc.

The demographic data has aspects that make the synthesis of data challenging. First of all, due to the amount of categorical features the data is sparse. Further, the data has so-called **structural zeros** [2]. Those are a combination of features that cannot exist at the same time. For example, a 9 year old boy cannot have a driving licence.

To provide better fitting and more detailed data, recent researches began to explore neural networks as an alternative to more conventional approaches. The Variational Autoencoders (VAE) [4] were proved to outperform methods such Gibbs Sampler and Bayesian Network when number of dimensions were high.

Another work [2], explored the ability of Wasserstein GAN (WGAN) [5] to achieve a similar performance (to VAE). It showed that while it does outperform many models in terms of the prediction power, it provides less diverse data in comparison to VAE.

Many new modifications to the GAN have emerged since, e.g. WGAN with Gradient Penalty [6] or Multi-Agent Diverse Generative Adversarial Networks (MADGAN) [7]. As well as new methods for approximation of the categorical distributions [2]

The goal of the project is to benchmark the recent research within population synthesis and GANs. It will allow to determine whether the new architectures can provide higher predictive power, as well as to produce high-fidelity data (i.e. with the possibility to increase the number of dimensions).

This project focuses on the implementation and optimisation of the following GAN architectures[3] with Gumbel Softmax [4]:

- WGAN with Gradient Penalty (GP-WGAN)
- Multi-Agent Diverse Generative Adversarial Networks (MADGAN)

The performance of GP-WGAN and MADGAN is compared to the WGAN, to see wheter the quality of data is improved.

Code might be found: https://github.com/carlomarxdk/GAN-Population-Synthesis

2. RELATED WORKS

The following section describes particular qualities of the GAN architectures that are used throughout the project.

[1] we is used for the flow of the text

[2] see Methodology, Gumbel Softmax

[3] see Related Works

[4] see Methodology

2.1. WGAN vs GAN

The *vanila* GANs might have a limited performance due to a weaker or limited convergence (i.e. mode collapse). According to the paper [5], the main issue is the **loss function**.

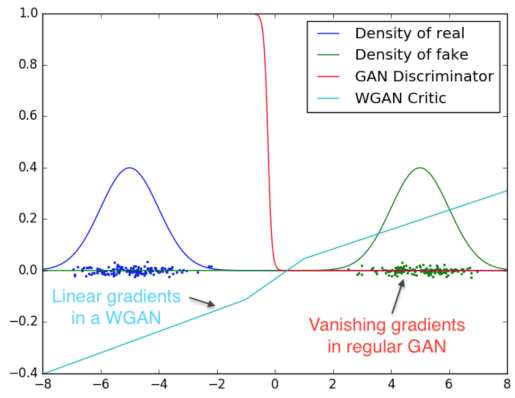

Fig. 1: Gradient provided by GAN's discriminator and WGAN's critic [5]

In the GAN setup, two networks (G and D) are acting against each other [2], trying to change the outcome of the following value function **V(D,G)**:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log\ D(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))] \quad (1)$$

In the Equation 1, **D** stands for a discriminator, **G** stands for a generator network, p_z is a noise (e.g. Gaussian) and p_{data} is a distribution of real data, $G(z)$ is a sample from a fake distribution given the input z, $D(G(z))$ is a discriminant estimate that output of $G(z)$ is real. The generator is only able to influence the $\log(1 - D(G(z)))$, thus, it tries to minimise that. At the same time, the discriminator network tries to maximise that term. Figure 1 shows that if the real distribution and the fake distribution (produced by generator) are far apart, then (in the vanilla GAN) the gradient vanishes and generator is incapable of learning anything.

On the other hand, the paper introduces the **Wasserstein Distance** as an alternative to the Minimax Equation (1). Figure 1 shows that WGAN loss is differentiable everywhere [5]. In that setup, we no longer ask a discriminator network to decide whether the sample is fake or real. Instead we are asking to give a score to the sample. So now the **critic** [5] is minimising the following function:

$$L_D = D(x) - D(G(z)) \quad (2)$$

And a generator is trying to minimise the following:

$$L_G = -D(G(z)) \quad (3)$$

[5]the discriminant is renamed to critic, as it's mechanism is altered

Not only it allows to overcome the issue of the vanishing gradient, but at the same time, the loss function provides a meaningful value, i.e. the quality of the generated data in relation to the original.

2.2. WGAN-GP

There is an issue with the WGAN: enforcement of the **Lipschitz constraint**. By utilising the Kantorovich-Rubinstein duality, the Wasserstein distance can be expressed as [5]:

$$W(p_r, p_g) = \frac{1}{K} \sup_{\|f\|_L \leq K} E_{x\sim p_r}[f(x)] - E_{x\sim p_g}[f(x)] \quad (4)$$

In the Equation 4, **f** is a 1-Lipschitz function (see Equation 5). In the WGAN, this constraint is enforced by the weight clipping (in the critic network) [5]. Thus, we make sure that weights of the critic are within some range $(-c, c)$ [6]

$$|f(x_1) - f(x_2)| \leq |x_1 - x_2| \quad (5)$$

However, the weight clipping (see Figure 2, right top) might lead to the edge case, when all the weights of the Critic Network are on either ends of the clipping range. This would lead to a unstable and long training (see Figure 2, left), which then might produce inflexible generator [6].

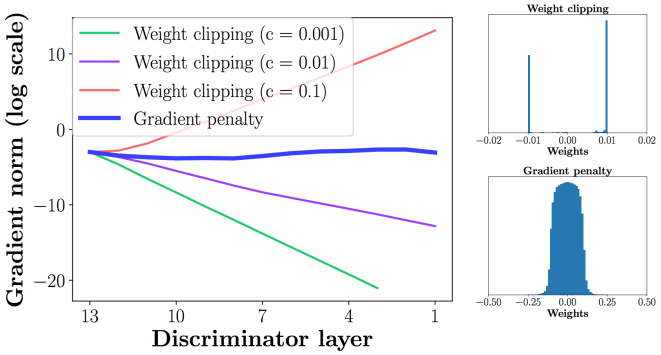

Fig. 2: Weights of the Critic: Weight Clipping vs Gradient Penalty [6]

The WGAN-GP introduces the **Gradient Penalty**, which provides a more diverse distribution of weights in the critic network (see Figure 2, right bottom). In order to do so, we need to take into consideration the following:*"A differentiable function is 1-Lipschtiz if and only if it has gradients with norm at most 1 everywhere"* [6]. The paper suggest to interpolate between $\tilde{\mathbf{x}}$ and $\mathbf{x}$ (i.e. real and generated data) in the Equation 6 , such that $\hat{x} = t\tilde{\mathbf{x}} + (1-t)\mathbf{x}$, where $t \in [0, 1]$.

Thus, if the norm of the gradient deviates from 1, the loss increases. The λ is usually set to 10.

[6]usually chosen to be around .1

$$L = \underbrace{\underset{\tilde{\boldsymbol{x}}\sim P_g}{E}[D(\tilde{\boldsymbol{x}})] - \underset{\boldsymbol{x}\sim P_r}{E}[D(\boldsymbol{x})]}_{\text{Original critic loss}} + \underbrace{\lambda_{\hat{\boldsymbol{x}}\sim P_{\hat{\boldsymbol{x}}}}\left[\left(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1\right)^2\right]}_{\text{Our gradient penalty}} \tag{6}$$

The WGAN-TP trains faster than WGAN, it also allows the Generator of the WGAN-TP to be more flexible and diverse when it comes to the data generation [6].

2.3. MADGAN

The Multi-Agent Diverse Generative Adversarial Networks [7] are designed to overcome the mode collapsing of the original GAN and better performance compared to WGAN. Figure 3, shows that MADGAN with 4 Generators fits better to the 1 Dimensional Gaussian Mixture Model than WGAN does.

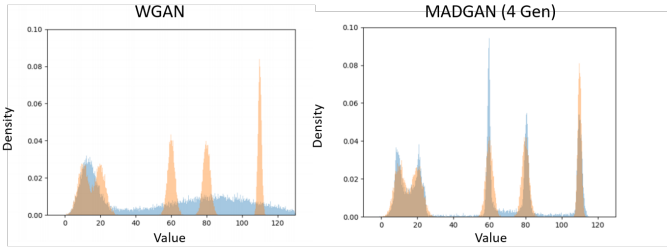

Fig. 3: MADGAN fitted to a mixture of Gaussian Distributions [7]

The basic idea of the MADGAN is to train multiple Generators and *push* them to learn different aspects of data. Figure 4 visualises the idea.

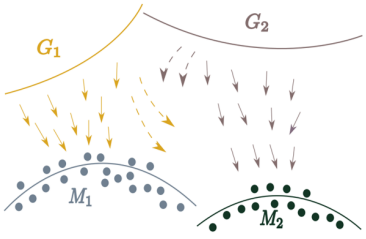

Fig. 4: Generators of MADGAN converging to different modes [7]

We ask discriminator to identify where the data comes from (e.g. is it from Generator 1, Generator N or is it a real sample) [7], so as to push each Generator to learn different aspects of data. Like in the GAN network we ask the discriminator to provide softmax scores, but now it should do it over $n+1$ classes (i.e. number of generators plus one class for real samples) [7]. Thus, instead of the Equation 1, now Discriminator maximises the following:

$$\max_D E_{x\sim p} H\left(\delta, D\left(x\right)\right) \tag{7}$$

In Equation 7, H tsands for the negative cross-entropy, δ stands for the Diract delta distribution $\delta \in \{0,1\}^{n+1}$ [7]. For $j \in \{1,\ldots,n\}$, the $\delta(j) = 1$ if it belongs to the j-th generator, otherwise if $\delta(n+1) = 1$ means that data comes from the real distribution [7]. Each j-th Generator minimises the same expression as in vanilla GAN.

3. DATA

The **Danish National Travel Survey (TU)**[8] was used for the model training and evaluation. The dataset documents the travel behaviour of the Danish Residents. The survey also includes the ones regarding the demographics of a person (e.g. age, annual income, family type, occupation, type of accommodation and many more). The data of $\approx$ 170k residents is represented in the dataset ($\approx 2.5\%$ of Danish Population).

3.1. Data Preprocessing

The dataset contains more than 70 different features. Due to the nature of dataset (i.e. survey), the data itself is noisy, some variables are missing responses and not all the features are relevant. The workflow for the preprocessing:

First step was to **handpick features**. Total amount of 50 features were chosen, including *number of a adults in a household*, *level of an education*, *possession of a driving license* etc.

Second step was to **manually infer certain features**. For example, the people who did not possess a driving license are missing the year of the obtaining the driving license (in the dataset). Thus, in all of these cases the year was set to -1.

Third step was to **drop the columns based on the missing rate**. Thus, if 20% of values were missing from a column, it was drooped from the dataset. This step reduced the amount of features to 33.

Fourth step was to **impute the missing values** using the K-Nearest Neighbours method. Crossvalidation showed that 2 neighbours imputed the missing values quite well. The *Fancy Impute package* [7] was used to perform the imputation.

Fifth step was to **normalise the numerical features**, such as year of birth, number of teens in a family etc. It was done using the **L2 Normalisation** [8].

Sixth step was to **drop variables based on the covariance**, i.e. if the correlation between the two is more than 95%. For example, the year of birth was dropped, as it was highly correlated with age and the year when the person obtain a driving license.

[7] https://pypi.org/project/fancyimpute/
[8] sklearn normalizer

Final step was to **bin the continuous numerical features**. Each of them were binned into 5 different categories. The final number of features in data was down to 21.

3.2. Data Representation

The preprocessed features are all categorical. All of the possible categories within each feature are mutually exclusive,and at the same time, most variables do not have any hierarchy when it comes to the categories (e.g. single vs married categories). Thus, it make sense to represent each variable using **one-hot encoding**. As the result, the **distribution of the data is 309-dimensional**, i.e. single person is described with a 309-dimensional vector of binary values.

4. METHODOLOGY

4.1. Generator Network

The structure of the generator network was similar for every GAN architecture: WGAN, WGAN-GP and MADGAN. It consisted of [9]:

1. Network inputs 100 values generated by the Gaussian Noise. The dense layer outputs 512 values; followed by the application of the LeakyReLU activation, the batch normalisation and dropout
2. Next dense layer outputs 1024 units consists of 1024 units, with LeakyReLU, batch normalisation and dropout.
3. Same happens with the 3rd and 4th layer (which have 1536 and 1024 units, accordingly)
4. The last Dense layer outputs 309-dimensional network, which is followed by the Gumbel Softmax (explained further).

4.1.1. Last Layer of the Generator

Due to the fact that the person data is represented as a concatenated vector of one-hot encoded features (see Figure 5), the activation function (e.g. softmax) should be applied separately on different parts of the tensor. For example, the softmax should be applied on the first and second "neuron" as those represent the gender; then, softmax should be applied separately on the neurons that represent education and so on.

Fig. 5: Representation of a person (abstract example)

[9] configurations of Generator can be found in Appendix A

4.1.2. Gumbel Softmax

If softmax is applied on the last layer of the generator, then the output would consist of probabilities. The Discriminator would then be able to distinguish between the real and generated data. Thus, it is required to transform the values into one-hot encoding.

The binarization (with argmax) of a tensor would **erase the gradient of the generator network**. In order to solve this, the Gumbel Softmax is used [9]. It provides a method to draw the one-hot encoded vectors from a categorical distribution [10]:

$$z = \frac{\exp((\log(\alpha_i) + g_i) \cdot \tau^{-1})}{\sum_{j=1}^{k} \exp((\log(\alpha_j) + g_j) \cdot \tau^{-1})} \tag{8}$$

In the Equation 8, $g = -log(-log(u))$, $u \sim \text{Uniform}(0, 1)$, meaning g is a sample from the standard Gumbel Distribution. The τ is so-called *temperature* constant. As $\tau \to 0$, the Gumbel Softmax approximation of the output becomes one hot [10] [10]. At the same time, the small values of τ would lead to the high variance of the gradient during the backpropagation, and vice versa [11].

This activation is used instead of the softmax activation on the last layer of the Generator Network (output is on the form: Figure 5).

4.2. Discriminator/Critic Network

The setup of the discriminator is the same except for the last layer [11]:

1. After the input layer (301-dimensional input), the linear layer follows with the 10245 units, leaky ReLU and Dropout.
2. Next linear layer has 512 units and it is also followed by Leaky ReLU and Dropout layer.
3. The following layer has 256 neurons, Leaky ReLU and Dropout
4. For the WGAN and WGAN-TP, the last layer consist of 1 unit (we are simply asking to assign the score). For the MADGAN, the last layer consists of 4 units (one for each of the three Generators and one for real data), followed by the softmax activation.

4.3. Experiment Setup

The **temperature for the Gumbel Softmax** was identical in all the setups: $1E^{-3}$. All the models are trained with the **RMSprop Optimizer** as the TensorFlow implementation in [2].

[10] i.e. max value will tend to be 1 and other will become 0

[11] see Appendix A for more information

For the **Wasserstein GAN**, the discriminator is changed for a critic network, and it is trained 5 times per epoch (while generator is trained just once). Meanwhile, the weights of the critic are clipped within the range of $(-.1, .1)$ after the back-propagation is complete. In **WGAN-GP** setup, the weight clipping is not performed, instead, the loss function includes the Gradient Penalty Term.

For the **MADGAN**, we train three separate generators and one discriminator network. The discriminator needs to decide which category the presented sample belongs to (e.g. was it 1st Generator etc.)

4.4. Evaluation of the Quality

In order to evaluate the quality of the synthesised data, the **partial empirical distributions** of both real and fake data samples is going to be [2] compared by using Standardised Mean Square Error (SRMSE), Pearson Correlation coefficient and coefficient of determination, R^2 [2]. '45^o-plots' are used to visualise the similarity.

4.5. Configurations

The models were trained on the Virtual Machine with the Cloud Deep Learning VM Image (Pytorch) [12]. It was hosted on the Google Cloud Platform. The VM had 8 vCPUs (Intel Skylake, 2.2 GHz), 30 GB of Memory and a GPU (NVIDIA Tesla P4). Python was used as a programming language.

5. RESULTS

The following section describes the performance of the models and the quality of the data produced.

5.1. Model Evaluation

On the Figure 6, for the WGAN: as the training progressed the Critic Loss slowly increased, the inverse trend could be observed for the Generator Network. According to the score, the distribution of real and fake data was coming closer, i.e. Critic Network gave a higher score to the fake data, meaning it had harder times to distinguish between the two distributions. On the last plot, the margin between real data and fake data got smaller.

On Figure 7, the discriminator loss for the MADGAN grew and all the generator losses decreased. However, in this case the loss function is not a good indicator of the data quality.

5.2. Quality of the synthesised data

The WGAN will serve as a baseline model, since we are to make a superior model [12]. The Table 1 shows the statistics

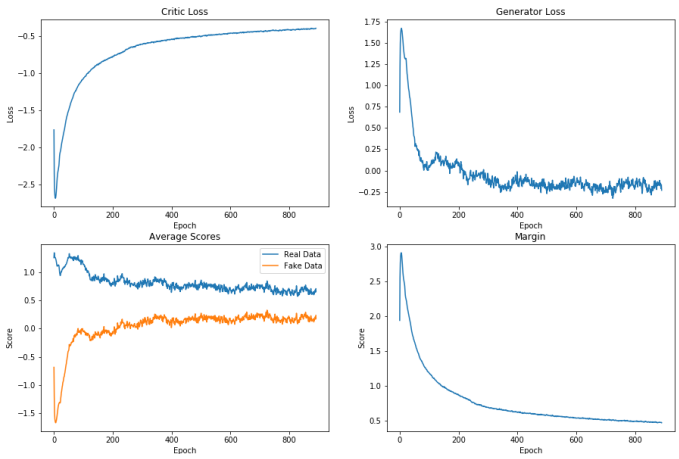

Fig. 6: Training of WGAN-GP: **Critic Loss** (left top) **Generator Loss** (right top) **Scores** that critic gave to real and fake data (left bottom) **Margin** between those scores (right bottom)

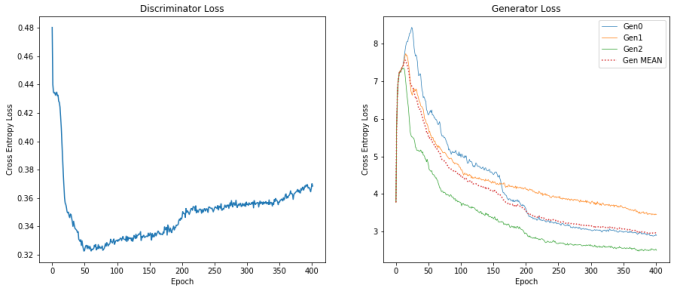

Fig. 7: Training of MADGAN: **Discriminator Loss** (left) **Multiple Generator Loss & Average Generator Loss** (right)

for the Partial Distributions of features (that can be combined in 1k different ways), the higher is the R^2 and correlation, the better is the model. MADGAN and WGAN provide similar performance, while WGAN shows slightly higher R^2, the MADGAN shows higher correlation and lower error. In this case, the WGAN-GP outperforms both models.

Table 2 shows the quality: when we increase the dimensionality (where the set of variables can form 128k different combinations). Here it is shown that MADGAN was performing much better than in a lower dimensional case, i.e. large increase in R^2 states that the fit is much better. At the same time, WGAN provided worse results (e.g. R^2 and correlation decreased, the error increased). WGAN-GP outperforms both models.

When we look into higher dimensional sets (e.g. when a possible combination of variables can reach to 6m or 100b) both WGAN and MADGAN fail (see Figure 8) [13]. WGAN and MADGAN did not capture any structure. WGAN-GP

[12] the statistics for the WGAN model can be seen in the Appendix B

[13] see Appendix C for the full statistics

Table 1: Quality of data: partial distribution of Age, Occupation and Driving Licence; the amount of possible combinations is around 1k

Model	R^2	Corr	SRMSE
WGAN	.51	.72	5.52
WGAN-GP	.99	.99	0.99
MADGAN	.48	.81	4.90

Table 2: Quality of data: partial distribution of Age, Education, Sex, Driving License, Bicycle Ownership and Occupation; total combinations 128k

Model	R^2	Corr	SRMSE
WGAN	.25	.50	7.12
WGAN-GP	.97	.98	1.42
MADGAN	.63	.86	3.93

were still able to provide good statistics. The axis of Figure 8 show the probability of the combination, in a high fidelity case (bottom right plot), the WGAN-GP is able to mimic the samples that have a small probability in the real dataset.

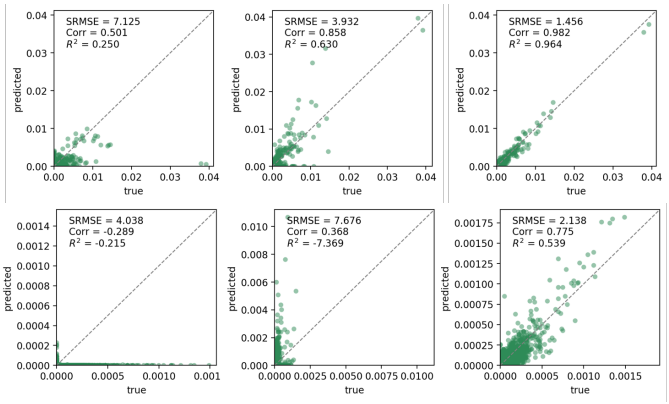

Fig. 8: 45-degree plot. Columns (left to right): WGAN, MADGAN, WGAN-GP. Rows (top to bottom): Set with 128k combinations, set with 100b combinations

6. DISCUSSION

6.1. WGAN

If we compare the performance of the WGAN with Gumbel Softmax and the TensorFlow implementation of WGAN [2] [14], we can see that it lowered the model performance. Original WGAN provides higher quality when the combination of features is around 1.5k (i.e. $R^2 = 98$ and $corr = 0.99$). For the evaluation in Table 1, we took similar features (as in the paper [2]), though not identical, as the preprocessing and data is slightly different; however, WGAN with Gumbel Softmax performs worse. One of the possible reasons is the additional of Gumbel Softmax. Meanwhile, the clipping value also influenced the training. In case of the clipping value equal to .01 or .05, the Generator of WGAN tends to converge after around 100 epochs (providing a poor data quality). As the final model was trained with the value 0.1, it did not lead to a fast convergence, instead the Generator loss rapidly increased at some point (see Appendix B).

[14] further referenced as Original WGAN

6.2. Gumbel Softmax

The introduction of Gumbel Softmax layer brought two issues. First, an epoch run time increased by the factor of 5 for each model. Secondly, the temperature constant can drastically change the training process. Lower values of the temperature did not allow gradient to flow through the Generator Network and higher values tend to produce representations that were away from the one-hot.

6.3. MADGAN Performance

The MADGAN was challenging to optimise. First of all, you need to decide how many generators to have. And even the writers of the paper [7] showed that the wrong number might lead to a drastically different result. More than that, during the training of the MADGAN, one edge case was discovered: the model was trained with 5 generators and slightly higher learning rate than it is now. At one point, all the generators started to produce identical samples, the discriminator loss increased as the discriminator had to randomly choose which generator produced the result. The generators were not able to learn anything even many epochs after. Lastly, the MADGAN was the slowest model to train. One MADGAN epoch was equal to approximately 4 WGAN-GP ones.

In all the cases, Adam Optimizer provided the worst training results (confirms the finding of the [2] for the WGAN). The models seem to converge very quick without providing good quality data. RMsprop optimizer provided better training.

7. CONCLUSION

Based on the result WGAN-GP with the Gumbel Softmax seem to be the best model. Even if we compare the result in Table 1 to the WGAN paper [2], the current implementation has better results. The model never finished learning even after 400 epochs, thus, with more time and resources it could have achieved even better results. More than that, it might be used to increase number of dimensions as the current model still can handle high-dimensional cases (e.g. to synthesise more diverse and detailed samples).

8. REFERENCES

[1] Stanislav S Borysov, Jeppe Rich, and Francisco C Pereira, "How to generate micro-agents? a deep generative modeling approach to population synthesis," *Transportation Research Part C: Emerging Technologies*, vol. 106, pp. 73–97, 2019.

[2] Sergio Garrido, Stanislav S Borysov, Francisco C Pereira, and Jeppe Rich, "Prediction of rare feature combinations in population synthesis: Application of deep generative modelling," *arXiv preprint arXiv:1909.07689*, 2019.

[3] Danqing Zhang, Junyu Cao, Sid Feygin, Dounan Tang, Zuo-Jun Max Shen, and Alexei Pozdnoukhov, "Connected population synthesis for transportation simulation," *Transportation Research Part C: Emerging Technologies*, vol. 103, pp. 1–16, 2019.

[4] Stanislav S Borysov, Jeppe Rich, and Francisco C Pereira, "Scalable population synthesis with deep generative modeling," *arXiv preprint arXiv:1808.06910*, 2018.

[5] Martin Arjovsky, Soumith Chintala, and Léon Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[6] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville, "Improved training of wasserstein gans," in *Advances in neural information processing systems*, 2017, pp. 5767–5777.

[7] Arnab Ghosh, Viveka Kulharia, Vinay P Namboodiri, Philip HS Torr, and Puneet K Dokania, "Multi-agent diverse generative adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8513–8521.

[8] Hjalmar Christiansen and Jonas Lohmann Elkjær Andersen, "Transportvaneundersøgelsen (tu): Dokumentation af spørgeskema anvendt til tu-interview i 2019," Mar 2019.

[9] Eric Jang, Shixiang Gu, and Ben Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[10] Yoel Zeldes, "Neural networks gone wild! they can sample from discrete distributions now!," Jul 2018.

[11] Li Xiucheng, "Gumbel-max trick," 2017.

[12] Google Cloud, "Creating a pytorch deep learning vm instance," 2019.