DATGAN: INTEGRATING EXPERT KNOWLEDGE INTO DEEPLEARNING FOR POPULATION SYNTHESIS

Gael Lederrey
Transport and Mobility Laboratory,
École Polytechnique Fédérale de Lausanne,
Lausanne, Switzerland
gael.lederrey@epfl.ch

Tim Hillel
Transport and Mobility Laboratory,
École Polytechnique Fédérale de Lausanne,
Lausanne, Switzerland
tim.hillel@epfl.ch

Michel Bierlaire
Transport and Mobility Laboratory,
École Polytechnique Fédérale de Lausanne,
Lausanne, Switzerland
michel.bierlaire@epfl.ch

September 1, 2021

Abstract

Agent-based simulations and activity-based models used to analyse nationwide transport networks require detailed synthetic populations. These applications are becoming more and more complex and thus require more precise synthetic data. However, standard statistical techniques such as Iterative Proportional Fitting (IPF) or Gibbs sampling fail to provide data with a high enough standard, *e.g.* these techniques fail to generate rare combinations of attributes, also known as sampling zeros in the literature. Researchers have, thus, been investigating new deep learning techniques such as Generative Adversarial Networks (GANs) for population synthesis. These methods have already shown great success in other fields. However, one fundamental limitation is that GANs are data-driven techniques, and it is thus not possible to integrate expert knowledge in the data generation process. This can lead to the following issues: lack of representativity in the generated data, the introduction of bias, and the possibility of overfitting the sample's noise.

To address these limitations, we present the Directed Acyclic Tabular GAN (DATGAN) to integrate expert knowledge in deep learning models for synthetic populations. This approach allows the interactions between variables to be specified explicitly using a Directed Acyclic Graph (DAG). The DAG is then converted to a network of modified Long Short-Term Memory (LSTM) cells. Two types of multi-input LSTM cells have been developed to allow such structure in the generator. The DATGAN is then tested on the Chicago travel survey dataset. We show that our model outperforms state-of-the-art methods on Machine Learning efficacy and statistical metrics.

STRC | 21st Swiss Transport Research Conference
Monte Verità / Ascona, September 12 – 14, 2021

Contents

1 Introduction

Population synthesis refers to combining different data sources to derive a representation of agents matching given criteria. Usually, it is compromised of methods that predict populations in social and geographic spaces. This area of research has seen increasing attention in recent years due to an increased focus on agent-based modelling in transportation (Miranda, 2019). In the past, data have been collected through phone surveys, household or individual travel diaries and questionnaires given by Census agencies. However, these surveys tend to cost much money. Therefore, much effort is made to reduce the size of such survey to cut the costs. Indeed, researchers are working on generating synthetic populations that represent the real population with as little data as possible. While reducing cost is one of the reasons why synthetic populations are needed, it is not the only one. For example, one might want to make sure that the collected data are anonymous. Using a synthetic population generator is a technique that has shown success over the years. In addition, being able to generate synthetic populations opens new opportunities for hypothetical scenarios. For example, we could derive information from one population to apply it to another one. The latter is called transfer learning. There are multiple reasons why synthetic population generation is needed, and researchers have come up with many different methods to generate them.

The three main existing approaches for generating synthetic populations are resampling techniques, simulation techniques and deep learning techniques. While the first two techniques have been well studied within the transport community, the latter comes from Machine Learning. Generative Adversarial Networks (GANs) are the main models used to generate synthetic data. While these models were first created to generate images, especially human portraits, researchers have developed new models to generate all kinds of data: language, time series, and tabular data (such as synthetic population).

In this article, we propose a novel model that controls the generation process of such a synthetic population. Indeed, while GANs have shown to generate accurate synthetic populations, the researchers have no control over the model. We, thus, propose to let the researcher design a network to represent the interactions between the variables with a Directed Acyclic Graph (DAG). This DAG is then used to generate the structure of the model that will generate such a population. Allowing researchers to control the process has two main advantages: they can tinker with the data generation process, create hypothetical populations, and control the dependencies for forecasting. In this article, we thus present our new GAN model named Directed Acyclic Table GAN (DATGAN). We show that it outperforms state-of-the-art synthetic data generators on multiple metrics. We also make a sensitivity analysis on the DAG to show its effect on the data generation process.

The rest of this article is laid as follows. In the next section, we present the Literature Review. We first introduce the existing approaches for population synthesis and then discuss the different research axes. Finally, we conclude the literature review with the opportunities and limitations of existing research. In Section 3, we present the whole methodology for the DATGAN. We discuss how to preprocess the data, what models are used for the generator and the discriminator, and how to use the DAG to create the generator's structure using LSTM cells. Section 4 presents the case study and Section 5 shows the results. We conclude this article in Section 6 and give ideas for future work.

2 Literature Review

As stated in Section 2.1, synthetic data generation is used to overcome five limitations that can be found in real datasets: simulation/activity-based modelling, Machine Learning efficacy, bias correction, privacy preservation, and transfer learning. These research axes are discussed in detail in Section 2.2. At the same time, methods for synthetic data generation can be grouped into three main approaches: resampling techniques based on Iterative Proportional Fitting (IPF), simulation techniques based on Markov Chain Monte-Carlo (MCMC), and deep learning approaches. The former two have been extensively researched in transportation and are usually motivated using activity-based models. The latter finds its origin in the Machine Learning community and are used to improve Machine Learning models. However, in recent years, transportation researchers have been using deep learning approaches for population synthesis whilst keeping the same motivation as previously. In Section 2.1, we give an overview of these three groups of methodologies. Finally, in Section 2.3, we discuss the opportunities and limitations of these techniques linked to the five research axes.

2.1 Existing approaches for population synthesis

Synthetic data have a large span of applications. For example, we can find such data in marketing, computer vision, security, and transportation. In the context of this article, we study the literature on one particular application: population synthesis. As stated in Section , population synthesis is included in the field of synthetic data generation, and it is mainly used for transportation-related applications. While the techniques for both of these fields are similar, this literature review only covers the approaches used for the latter.

2.1.1 Resampling techniques

The first method used for generating synthetic populations is the Iterative Proportional Fitting method (IPF) presented by Beckman et al. (1996). The method was first introduced by Deming and Stephan (1940). It consists of proportionally adjusting a matrix to produce a new table such that the specified marginals are individually conserved. In other terms, the idea is to memorize the marginal (sum of all the elements) of the columns and the rows in a given dataset. The algorithm starts by filling the synthetic dataset with random values. Then, it iterates over the columns and the rows such that the sum becomes equal to the marginal of the original dataset. Therefore, such methodology does not have any interaction between the rows nor the columns. In addition, the type of values is not always respected, as shown in Figure 1.

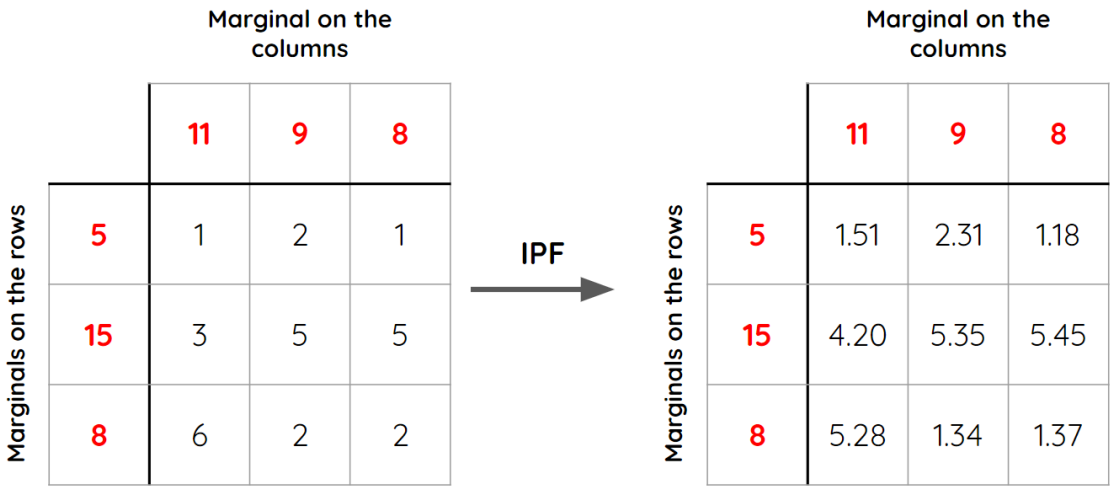


Figure 1: Example of a dataset transformation using the IPF methodology. The table on the left is the original table and the one on the right is the synthetic table.

Beckman et al. (1996) use this methodology to create a synthetic population based on the SF3 (San Francisco area) census data. Auld et al. (2009) and Barthelemy and Toint (2013) both propose to improve the IPF methodology using a multi-step procedures. For example, Barthelemy and Toint (2013) propose to separate the generation of individuals and households. They first generate a pool of individuals and then estimate the households' joint distributions. Finally, they gather individuals inside households to generate the synthetic population. They show that their methodology outperforms the standard IPF approach on a synthetic Belgian population using the Absolute Percentage Difference (APD) and the Freeman-Tukey Goodness-of-Fit test. Rich (2018) improved the IPF method even further using a three-step procedure to generate a synthetic population. He first pre-processes the data by harmonizing the constraints. He then uses IPF for the matrix fitting procedure for the individuals and finishes his procedure with a micro-simulation on the households.

While IPF methods are simple to implement, there are multiple majors issues with this technique. The first one is that there is no interaction between the variables with the basic algorithm. It is possible to add these interactions by adding multiple dimensions to the table. However, for each level of interaction, one more dimension has to be added to the table. It, thus, quickly become a computationally expensive algorithm. In addition, IPF cannot differentiate between structural and sampling zeros. Multiple methods have been suggested to avoid sampling zero issues in the literature, such as Auld et al. (2009). Finally, IPF cannot differentiate between the different types of data (categorical and continuous). Thus, researchers have been developing new techniques to generate synthetic populations, such as MCMC simulation.

2.1.2 Simulation techniques

The first use of simulation for generating synthetic population dates from 2013. Farooq et al. (2013) proposes to use a Markov Chain Monte Carlo (MCMC) simulation using Gibbs sampling to generate synthetic populations. They show that this simulation technique outperforms IPF methods using multiple statistical metrics such as R^2 and Standardized Root Mean Squared Error (SRMSE) (Müller and Axhausen, 2010). Casati et al. (2015) improved on the MCMC approach of Farooq et al. (2013) by adding a hierarchical structure to the simulation process. They rank the individuals living in the same household according to their role. Kim and Lee (2016) and Philips et al. (2017) both propose a mix

of Simulated Annealing (SA) and simulation to generate the synthetic population. In the case of Kim and Lee (2016), the SA algorithm is used to generate potential individuals, while the MCMC simulation is used to decided whether to select or dismiss the distribution to avoid the hill-climbing phenomenon.

2.1.3 Deep learning techniques

Recent advances in deep learning and data generation have enabled new approaches for generating synthetic populations. For example, Borysov et al. (2019) use a Variational AutoEncoder (VAE) (Kingma and Welling, 2014) to generate synthetic population. VAE aim to reduce the dimensionality of the data into an encoded vector in the latent space. Data can then be generated more easily in this latent space since it is smaller in dimensionality. Borysov et al. (2019) demonstrated that their VAE model outperforms both IPF and Gibbs sampling for generating complex data. Another deep learning approach to generate synthetic data are Generative Adversarial Networks (GANs) introduced by Goodfellow et al. (2014). The key concept of the GAN is to train two neural networks against each other: a *generator* and a *discriminator*. The generator processes random noise to produce synthetic data. The discriminator (or critic) then evaluates the synthetic data against real data to provide a classification or continuous score on each data point on whether the data is real or synthetic. Initial GANs made use of a binary classifier for the discriminator network. However, Arjovsky et al. (2017) demonstrated that the use of a discrete loss function results in issues such as vanishing gradients. They thus propose an alternative continuous loss function based on the Wasserstein distance. This GAN is therefore named Wasserstein GAN (WGAN). Further key developments in GAN research include the introduction of a penalty on the gradient during model training (Gulrajani et al., 2017) or the addition of conditionality (Mirza and Osindero, 2014). Whilst the primary application of GANs has been the generation of image data, with a particular focus on human faces (Alqahtani et al., 2021), researchers have also developed specific architectures for tabular data. Such GANs can thus be used to generate synthetic populations.

For example, TableGAN (Park et al., 2018) and TGAN (Xu and Veeramachaneni, 2018) are two specific GANs models for tabular data. TableGAN has been developed with privacy-preservation techniques in mind. This model is based on Deep Convolutional GAN (DCGAN) (Radford et al., 2016). On the other, TGAN has been developed to reproduce tabular data as realistically as possible using Long Short Term Memory (LSTM) cells for the generator (Hochreiter and Schmidhuber, 1997). The authors demonstrated that the TGAN outperforms TableGAN when both models are used to generate synthetic data. In the transport community, researchers have also developed their own GAN structures to generate synthetic populations. For example, Garrido et al. (2019) developed their own GAN structure based on the WGAN to use tabular data. They showed that this new model was statistically better than IPF techniques, Gibbs sampling and the VAE of Borysov et al. (2019). Finally, Badu-Marfo et al. (2020) created a new GAN named Composite Travel GAN (CTGAN). Their GAN is based on the Coupled GAN (CoGAN) (Liu and Tuzel, 2016) and is used to generate the table of attributes for the population and the sequence of Origin-Destination segments. They show that the CTGAN outperforms the VAE statistically.

2.2 Research axes

The previous section shows that the primary focus of existing population synthesis in the transportation domain has been for direct use in simulation models. On the other hand, the deep learning community motivates their research by stating that using more data improves the efficacy of Machine Learning models. For example, Jha et al. (2019) show that a more extensive dataset leads to better validation and fewer uncertainties. Other examples discussing the dataset size can be found in the literature (Barbedo, 2018, Linjordet and Balog, 2019). However, this does not represent the only research axis for population synthesis. In the remainder of this section, we present and discuss five different directions for the research in data generation.

2.2.1 Simulation/activity-based modelling

Activity-based models, as explained by Bhat and Koppelman (2003), are a type of model used to understand travel demand using a behavioural approach. It is often linked to trip-based models. However, the latter tend to only look at the trips without taking into account humans' behaviour. Activity-based models, thus, require datasets with multiple information on individuals, such as socio-economic characteristics and trips.

In Section 2.1, we already presented multiple articles in the transportation community discussing data generation/population synthesis. Indeed, most of these articles' motivation is about the creation of synthetic population such that they can be used for activity-based models or simulation. For example, Farooq et al. (2013) developed a simulation technique based on Gibbs sampling for population synthesis. They state that their research is motivated using the idea that activity-based modelling requires synthetic population and demographic updates. More recently, Borysov et al. (2019) and Garrido et al. (2019) both motivate their article by saying that population synthesis has received increasing

attention due to an increased focus on agent-based models (ABMs). These models, for example, are used to simulate the behaviour of individuals, or group of individuals, to assess their effect on the whole system. Similar data are required for both ABMs and activity-based models.

2.2.2 Privacy Preservation

Privacy preservation techniques consist of ensuring that any private information is not disclosed while using data or Machine Learning models. For example, trip diaries datasets with precise origin and destinations might give out too much information and lead to malicious use of the data. Usually, privacy preservation already starts at the data mining step. However, it might not always be possible. Thus, data needs to be altered to avoid leaking any private data.

While privacy preservation has received much attention recently, it has always been used to motivate synthetic population generation research. For example, Barthelemy and Toint (2013) present a three-step procedure using IPF (Iterative Proportional Fitting) to improve the privacy preservation of the standard IPF methods. They state that the standard method tends to repeat observations, and thus it is possible to retrieve information from the true dataset. More recently, Park et al. (2018) developed the table-GAN, which has been specifically designed to preserve the original datasets' privacy. They show that their synthetic dataset leads to similar Machine Learning efficacy. However, they do not try to use data augmentation to show improvements in the Machine Learning models. In computer vision, multiple GANs models have been created with privacy preservation as the core motivation. For example, Liu et al. (2019) created the Privacy-Preserving GAN (PPGAN). This GAN uses differential privacy by adding noise that has been specifically designed in their case to the gradient during the learning procedure. Yin and Yang (2018), on the other hand, directly generated protected data within the generator of their GAN by removing some sensible information and encoding them in the generated data. They tested their synthetic data against attack models to show that their GAN can generate more complex data to be deciphered.

2.2.3 Machine Learning Efficacy

Machine Learning efficacy corresponds to study how efficient and why are Machine Learning models. Usually, it is done using prediction as to the core component of the studies. As stated in the introduction of Section 2.2, multiple articles have been published on the impact of the dataset size on the Machine Learning models. It is known that having larger datasets will often lead to better results in prediction. Thus, researchers have been working on finding ways to make existing datasets larger. It is called data augmentation. This concept is already widely used on images (Shorten and Khoshgoftaar, 2019). While simple techniques such as rotating or scaling images can be used in Computer Vision, it is impossible to apply such simple tricks on tabular data. Thus, researchers have been developing models aiming at augmenting tabular data.

Since GANs are one of the most promising research axes on data augmentation, multiple researchers have been working on this topic. However, as stated in Section **??**, most of GANs applications are about Computer Vision. Nonetheless, we can still find a few articles dealing with tabular data. Among these articles, Xu and Veeramachaneni (2018) motivates the development of the TGAN because organisations are using Machine Learning on relational tabular data to augment process workflows carried out by humans. They say that these synthetic datasets can either be used as an augmentation for the existing datasets or as a mean to preserve privacy, which is discussed in Section 2.2.2. The follow-up article of Xu et al. (2019) does not give a clear motivation on the usage of synthetic datasets. However, they test their models on Machine Learning efficacy by replacing the training data with the generated synthetic data.

2.2.4 Bias Correction

Bias correction is used when a dataset is not homogeneous. The collected data often do not reflect the true distributions. For example, if one might create a travel survey using smartphones, there is a high possibility of getting fewer answers from the elderly than the youth. Therefore, such datasets have to be corrected according to some known statistics. In Machine Learning, we often talk about an imbalanced dataset in which one or multiple classes are under/over-represented. Usually, standard techniques rely on sampling methods (Rubin, 1973) to rebalance the dataset. However, data generation techniques can also be used to augment the dataset and rebalance it.

If one has to rebalance a dataset with synthetic data, the easy path is to "hand-pick" the synthetic data close to the missing data from the original dataset and add them to the original dataset. While this technique might work, it requires the generation of many data. Indeed, synthetic datasets are generated to be as close as the original dataset. Thus, a category underrepresented in the original data will also be underrepresented in the synthetic dataset. Therefore, Machine Learning researchers have developed the Conditional GANs (Mirza and Osindero, 2014), an updated version of the GAN that allows generating data based on their labels. This, thus, increases the probability to generate synthetic data with the given label. Xu et al. (2019) have adapted this methodology to tabular data with the Conditional Table GAN

(CTGAN). They show that the conditionality is especially efficient for Machine Learning models when the data is highly imbalanced. They created synthetic datasets addressing the imbalance and trained Machine Learning models on both these datasets and the original dataset. The models trained on the synthetic datasets performed better than the ones trained on the original dataset. Previously, Farooq et al. (2013) motivates their research on population synthesis with Gibbs sampling using the fact that it can complete datasets.

2.2.5 Transfer Learning

Transferability, or transfer learning in Machine Learning, represents the fact to train a model on a dataset and then apply this knowledge to another, usually smaller or incomplete, dataset. Multiple GANs models have been developed in the past couple of year using transfer learning for images. However, the main purpose of using transfer learning is to reduce the computational cost. Indeed, training GANs on large datasets takes a lot of time and resources. Noguchi and Harada (2019) proposed a new method using the BigGANto transfer the knowledge learned on large datasets and apply this knowledge to a dataset with only 25 images. They show that they are able to add a new class to a pre-trained generator without disturbing the performance on the original domain. Wang et al. (2020) proposed to use a miner network that identifies which distribution of multiple pre-trained GANs is the most beneficial for a specific target. This mining pushed the sampling towards more suitable regions in the latent space. The MineGAN is therefore able to transfer the knowledge of multiple GANs such as the BigGAN and the Progressive GAN to a domain with fewer images. Jeon et al. (2020) takes a slightly different approach. Indeed, the goal of this research is to develop an image detection framework to classify between real and GANs-generated images. The T-GD is using both a teacher and a student network that works in pair for the detection. The teacher model is pre-trained on source data while the student model is pre-trained on target data. This framework allows the student network to require only a small amount of data to be able to efficiently detect if an image is real or has been generated. Frégier and Gouray (2020) propose another transfer learning method that consists in freezing the low-level layers of both the critic and the generator of the original GAN. Indeed, they reuse the weights of an autoencoder already trained on a source dataset. The weights of the low-level layers will be given to the decoder and encoder. The MindGAN is, therefore, a subnetwork trained as a GAN on the encoded features of the target dataset.

2.3 Opportunities and limitations

As we have discussed in Section 2.1, deep learning models are promising in multiple aspects. First, they outperform previous models in terms of statistics, as has been shown multiple times in the transportation community (Borysov et al., 2019, Badu-Marfo et al., 2020). In addition, researchers have already shown that their models can improve Machine Learning efficacy and bias correction. Furthermore, transfer learning is a promising research axis, but it has not been explored in the transportation community yet. Finally, the use of synthetic population for simulation has been discussed multiple times already. However, the generation of such a population raises multiple limitations with the current available deep learning models.

For example, the deep learning community have developed their own performance metric, similarly for the transportation community. However, we cannot find any standard practice to evaluate the generation of synthetic populations yet. While the goal of generating synthetic data might be different, a standard assessment method should be applied across these fields. For example, we lack a metric to assess how representative a synthetic population is compared to the original dataset. In addition, we know that Machine Learning models are usually prone to overfitting. We thus need to have a better assessment method. Finally, one of the main issues with deep learning models is the lack of control over the generation process. Indeed, neural networks are considered as black-box models in which humans do not interfere. Since these models have been initially created to improve the prediction power of Machine Learning models, they excel at creating a new population for this purpose. However, we state that integrating expert knowledge with these models helps us improve the quality of synthetic populations, especially in simulation. For example, the variables of a dataset do not make sense for a neural network. It will only learn from the data. However, an expert can help the neural network if they can provide the structure of the variables. In this work, we aim at delivering such a model.

3 Methodology

The Directed Acyclic Graph Tabular GAN (DATGAN) is a direct extension of the Tabular GAN (TGAN) published by Xu and Veeramachaneni (2018). Indeed, we kept the same pre-processing, described in Section 3.1, as well as the discriminator, in Section 3.3. On the other hand, we modified the generator of the TGAN, see Section 3.2, to support the use of a directed acyclic graph to model the relationship between the different variables in a given dataset.

Before introducing every component of the DATGAN, we need to define some mathematical elements. We define the table containing the original tabular data as $\mathbf{T}$. It contains n_c continuous random variables $\{C_1, \ldots, C_{n_c}\}$, and n_d

discrete random variables $\{D_1, \ldots, D_{n_d}\}$. The generative model $\mathbb{M}\left(C_{1;n_c}, D_{1;n_d}\right)$ is used to learn the unknown joint distribution $\mathbb{P}\left(C_{1;n_c}, D_{1;n_d}\right)$ and create a synthetic dataset named $\mathbf{T_{synth}}$. In this case, we do not consider sequential data. Therefore, each row is sampled independently from the joint distributed and is denoted using lowercase characters $\{c_{1,j}, \ldots, c_{n_c,j}, d_{1,j}, \ldots, d_{n_d,j}\}$.

3.1 Data preprocessing

In tabular data, we generally encounter two types of variables: continuous and categorical. Continuous variables can be drawn from complex multimodal distributions, in which categorical variables are drawn from a finite set of unique values. However, Neural Networks are not able to generate such complex data without some preprocessing. Indeed, Neural Networks generally works better in the range $(-1, 1)$. We, thus, transform the continuous variables into two variables using Gaussian Mixture Models (GMMs) and the categorical variables into a multinomial distribution.

3.1.1 Continuous variables

Xu and Veeramachaneni (2018) selected three datasets for their case study. They show that in two out of the three, continuous variables are, in the majority, multimodal. It is, therefore, mandatory to take into account this multimodality. As stated by Xu and Veeramachaneni (2018), using normalization and $\tanh$ function would make the gradient saturate when back-propagating if a mode is close to -1 or 1, we use the same methodology as the authors to transform the continuous variables. The idea is to cluster the values of the continuous variables using a Gaussian Mixture Model (GMM). We do this for both uni- and multimodal variables. Indeed, if a variable has only one mode, the GMM returns a very low probability to $m - 1$ components and only works with one component.

We want to transform each continuous variable $c_{i,j}$ into two different variables: a vector of probabilities $\mathbf{u}_{i,j}$ and the scalar value $v_{i,j}$. To achieve this, we first train a GMM with $m = 5$ components. The means and standard deviations are $\left\{\eta_i^1, \ldots, \eta_i^m\right\}$ and $\left\{\sigma_i^1, \ldots, \sigma_i^m\right\}$. These values are not known by any components of the DATGAN. We can then compute the probability $u_{i,j}^k$ that $c_{i,j}$ belongs to the Gaussian distribution k for each $k = 1, \ldots, m$. These probabilities are stored in the vector $\mathbf{u}_{i,j}$. Finally, we normalize $c_{i,j}$ as

$$v_{i,j} = \frac{c_{i,j} - \eta_i^{k^*}}{2\sigma_i^{k^*}} \tag{1}$$

where $k^* = \arg\max_k u_{i,j}^k$. For numerical purposes, we clip $v_{i,j}$ to $[-0.99, 0.99]$.

The postprocessing for continuous variable is straighforward. Indeed, we can get $c_{i,j}$ from $\mathbf{u}_{i,j}$ and $v_{i,j}$ using the following formula:

$$c_{i,j} = 2v_{i,j}\sigma_i^{k^*} + \eta_i^{k^*} \tag{2}$$

where $k^* = \arg\max_k u_{i,j}^k$.

3.1.2 Categorical variables

The difficulty with categorical variables is to make the model differentiable. In our case, categorical variables tend to have few unique values. Therefore, it is possible to generate a probability distribution using softmax. However, it is necessary first to transform these variables into one-hot encoding representation. We also add some noise to smooth the results.

Therefore, the first step is to transform each categorical variable $d_{i,j}$ in a $|D_i|$-dimensional one-hot vector $\mathbf{d}_{i,j}$. We then add Gaussian noise to each dimension of $\mathbf{d}_{i,j}$, thus giving:

$$\left(\widehat{\mathbf{d}}_{i,j}\right)_k = \left(\mathbf{d}_{i,j}\right)_k + \mathcal{N}(0, \gamma) \tag{3}$$

where $\gamma = 0.2$. We can normalize the final representation to get:

$$\widetilde{\mathbf{d}}_{i,j} = \frac{\widehat{\mathbf{d}}_{i,j}}{\sum_{k=1}^{|D_i|}\left(\widehat{\mathbf{d}}_{i,j}\right)_k} \tag{4}$$

After preprocessing of both the continuous and the categorical variables, we obtain a new table $\widehat{\mathbf{T}}$ that has now $n_c(m + 1) + \sum_{i=1}^{n_d} |D_i|$ variables. Each row of $\widehat{\mathbf{T}}$ is represented as $\left\{v_{1,j}, \mathbf{u}_{1,j}, v_{2,j}, \mathbf{u}_{2,j}, \ldots, v_{n_c,j}, \mathbf{u}_{n_c,j}, \widetilde{\mathbf{d}}_{1,j}, \ldots, \widetilde{\mathbf{d}}_{n_d,j}\right\}$. This vector is the output of the generator as well as the input of the disciminator.

3.2 Generator

The role of the generator $\mathbf{G}$ is to generate synthetic data to fool the discriminator. We use Long-Short Term Memory (LSTM) network to generate the variables in $\widehat{\mathbf{T}}$. Usually, the LSTM cells are arranged linearly following a given order. In the case of the TGAN, the LSTM cells follow the order of the columns in the dataset. For example, if the dataset looks as in Figure 2a, the LSTM cells will have the same order, as shown in Figure 2b. However, multiple problems can arise from this configuration. In the case of Figure 4, we have multiple issues with the configuration of the TGAN:

- If the variable "Mode choice" is the variable that we want to predict, it should, logically, inherit all the information from the other variables. In the case of the TGAN, the order of the columns can be rearranged to fix this.
- Some variables might not be useful to generate the others. In the case of Figure 4, the variable "Type of survey" should not be used to generate the others since it should not influence the mode choice. In the case of TGAN, it is possible to remove it from the dataset completely. However, if this variable were used for other purposes, it would not appear in the synthetic data.
- While LSTM has a long term memory, information tends to fade away the more cells there are. Therefore, we state that some variables should directly influence others. For example, both "Work status" and "Age" will influence the ownership of a driving license. Therefore, we want these two variables to directly pass information to the LSTM cell to generate the variable "Driving License". However, this creates an issue since an LSTM cell cannot take multiple inputs as is. We discuss this in Section 3.2.1.

While the first two issues can be tackled by changing the input data in the TGAN, the DATGAN aims to fix all of these issues simultaneously using a DAG to represent the relationships between the variables. The DAG can either be created by experts or using any statistical/ML technique using the data. In the context of this article, we use expert knowledge to create it. We leave the automatic specification of the DAG for future improvements. In Figure 2c, we show a possible DAG for the mock dataset in Figure 2a. In this case, we see that the variables "Age" and "Work status" are both directly linked to the variable "Driving license". Usually, LSTM cells cannot take multiple inputs. To tackle this issue, we provide two different solutions in Section 3.2.1. For all the other variables without multi-input, we keep the same structure as done in the TGAN.

The DATGAN generates continuous variables in 2 steps. The scalar value v_i is first generated, followed by the vector of probabilities $\mathbf{u}_i$. For both steps, a single LSTM cell is used. The categorical variable is generated in only one step. Figure 3 shows how information is passed between two LSTM cells. The elongated blue hexagons represent the variables used in the LSTM cells. The grey rounded rectangles represent Neural Networks operators such as the LSTM cells, Fully Connected Layers (FCC), or other functions performed by `tensorflow`. Finally, the orange ellipses represent mathematical operators. In Figure 3, we simplify the representation of an LSTM cell by only showing the inputs and outputs of such a cell[1].

First, we need to define the following sizes:

- $\mathcal{H}$: Size of the hidden state of an LSTM unit. This value is generally chosen based on the hardware and the complexity of the task.
- $\mathcal{B}$: Size of the input batch. Inputs are very rarely fed one by one. They are usually fed into any LSTM based model in the form of a subset of the total number of examples, *i.e.* batch.
- $\mathcal{D}$: Size of the inputs, i.e. number of features in the input data. In this case, it is a variable with random noise.

LSTM cells take two different variables as inputs: the cell state and the input variable. The input variable of the LSTM cell $\texttt{LSTM}_t$ is a concatenation ($+\!\!+$) of three variables:

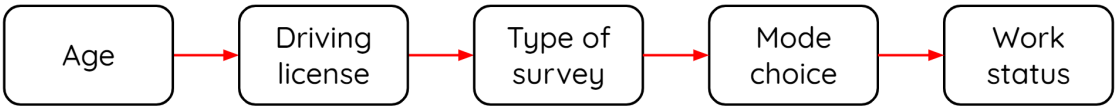
- $z \in \mathbb{R}^{\mathcal{B}\times\mathcal{D}}$: a random variable such that each dimension is sampled from $\mathcal{N}(0,1)$.
- $f_{t-1} \in \mathbb{R}^{\mathcal{B}\times\mathcal{H}}$: the output of the previous LSTM cell (or an embedding vector depending in the case of a continuous variable)
- $a_{t-1} \in \mathbb{R}^{\mathcal{B}\times\mathcal{H}}$ ($\texttt{att}_{t-1}$ in Figure 3): a weighted context vector, also name the attention vector. It is built using information from all the ancestors of the current LSTM cell.

The concatenation is done on the second dimension. Therefore, the final input has the size $\mathcal{B} \times (2\mathcal{H} + \mathcal{D})$. The cell state $C_{t-1} \in \mathbb{R}^{\mathcal{B}\times\mathcal{H}}$ is also given to the LSTM cell. If an LSTM cell does not have an ancestor, f_0 is learned during

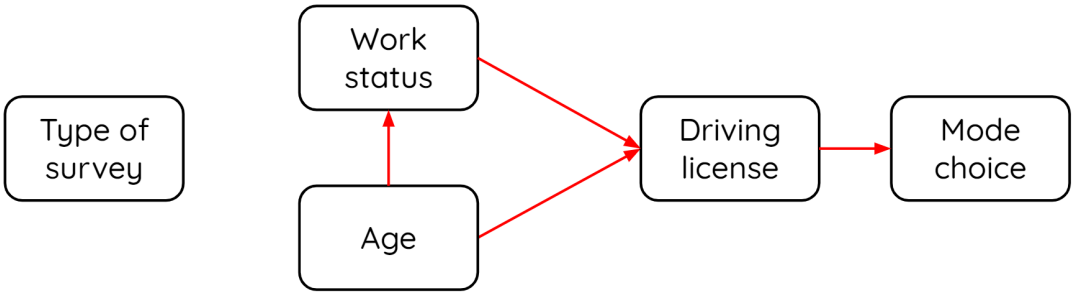
[1] We refer the reader to https://colah.github.io/posts/2015-08-Understanding-LSTMs/ for a detailed explanation of how an LSTM cell works.

Age	Driving license	Type of survey	Mode choice	Work status
continuous	categorical	categorical	categorical	categorical
0~100	"Yes" or "No"	Internet Phone ...	Driving Soft mobility ...	Employed Retired ...

(a) Example of a mock dataset



(b) Structure of the LSTM cells in the TGAN



(c) Structure of the LSTM cells in the DATGAN

Figure 2: Example of the structure of the data. Figure (a) shows the structure of a table with five variables. Figure (b) and (c) show the structure of the LSTM cells in the generator of the TGAN, respectively, the DATGAN.

the learning process while a_0 is a zero vector and C_0 is the zero state vector from `tensorflow`. The LSTM cell then provides two variables as outputs: the new cell state $C_i \in \mathbb{R}^{\mathcal{B}\times\mathcal{H}}$ and the output $o_i \in \mathbb{R}^{\mathcal{B}\times\mathcal{H}}$. The cell state C_t is directly passed to the new LSTM cell as the previous cell state, and it is stored ($\frown$) in a list of cell states $\mathbf{C}$. However, we cannot pass the output of $\texttt{LSTM}_t$ to $\texttt{LSTM}_{t+1}$ as is since we are using an attention vector to keep a better long-term memory. Indeed, the output o_i is transformed through multiple Fully Connected Layers (FCC) to change its size according to the desired variable. The first one uses an FCC with a $\tanh$ activation function to make sure the output values are within the specified bounds. It leads to $h_i \in \mathbb{R}^{\mathcal{B}\times\mathcal{H}}$. From h_i, we can produce the different outputs:

- if $w_t = v_t$, the value part of a continuous variable, we have $v_i = \tanh(W_{w,t}h_t)$ and $f_t = h_t$.
- if $w_t = u_t$, the cluster part of a continuous variable, we have $u_i = \text{softmax}(W_{w,t}h_t)$ and $f_t = \mathbb{I}(W_{f,t}u_t)$.
- if $w_t = \widetilde{\mathbf{d}}_t$, the output of a categorical variable, we have $\widetilde{\mathbf{d}}_t = \text{softmax}(W_{w,t}h_t)$ and $f_t = E_i\left[\arg\max \widetilde{\mathbf{d}}_t\right]$, where $E_i \in \mathbb{R}^{|D_i|\times\mathcal{H}}$ is an embedding matrix for the categorical variable D_i.

f_t is then concatenated with the random variable z before concatenating both with a_t as explained earlier. The variable a_t corresponds to the attention vector for the cell $\texttt{LSTM}_{t+1}$. In order to compute it, we need to learn the variable $\alpha_t \in \mathbb{R}^{|A_{t+1}|}$ (`attw` in Figure 3) where A_{t+1} corresponds to the list of ancestors of the current LSTM cell, $\texttt{LSTM}_{t+1}$. Using the functions `softmax`, `stack`, and `reduce_sum` in `tensorflow`, we are able to compute the new attention vector:

$$a_t = \sum_{k\in A_{t+1}} \frac{\exp \alpha_{t,k}}{\sum_j \exp \alpha_{t,j}} (\mathbf{C})_k \tag{5}$$

At this point, we know how to generate variables and to pass information from one LSTM cell to another. In the case of the TGAN that uses a linear representation of the dependencies between the variables, see Figure 2b, we have enough information to build the generator. However, in the case of the DATGAN using a DAG to represent the variables' dependencies, see Figure 2c, we need to address two issues:

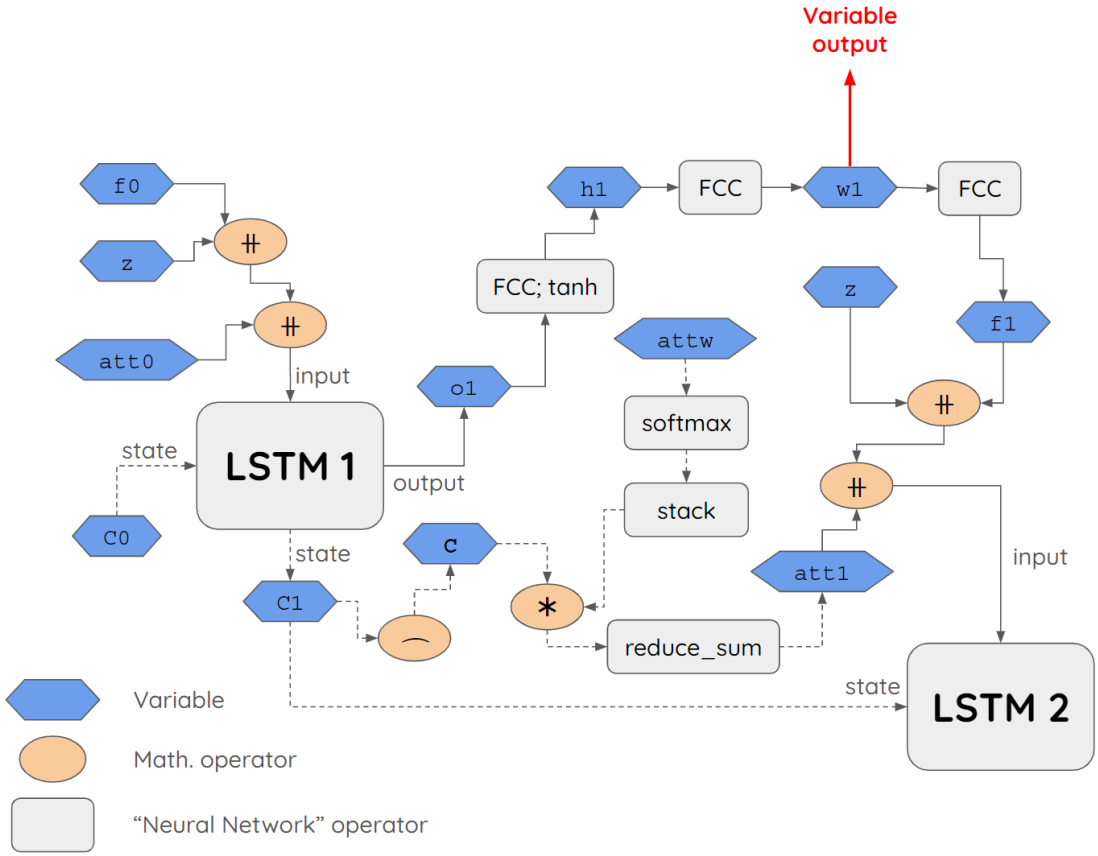


Figure 3: Representation of the mathematical operators, "Neural Network" operators and variables between two LSTM cells in the DATGAN.

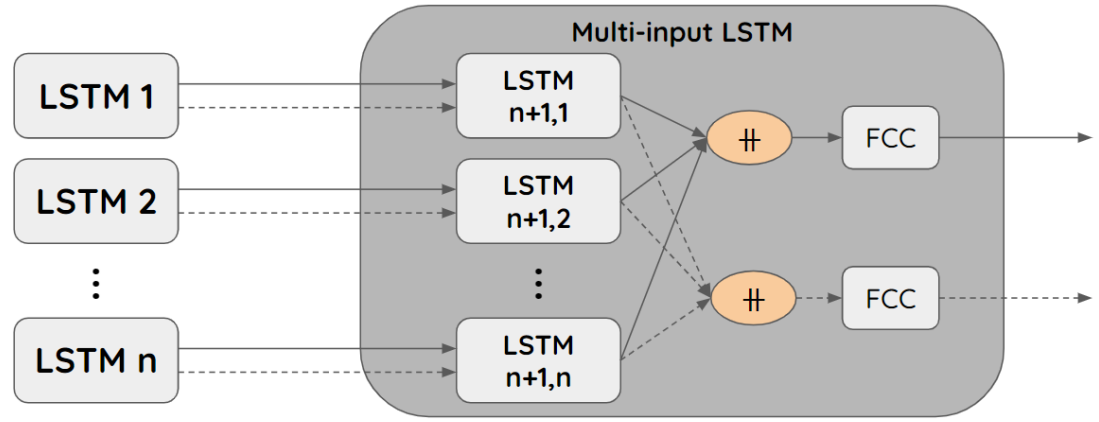
- How can we create an LSTM cell that takes multiple other LSTM cells as inputs? Indeed, Figure 3 only provides the answer for a single LSTM cell as an input. We discuss the solutions to this problem in Section 3.2.1.
- How can we build the sequence of LSTM cells using the DAG? We have to be careful about multiple elements while using the DAG to create the generator. We, thus, discuss this in Section 3.2.2.

3.2.1 multi-input LSTM

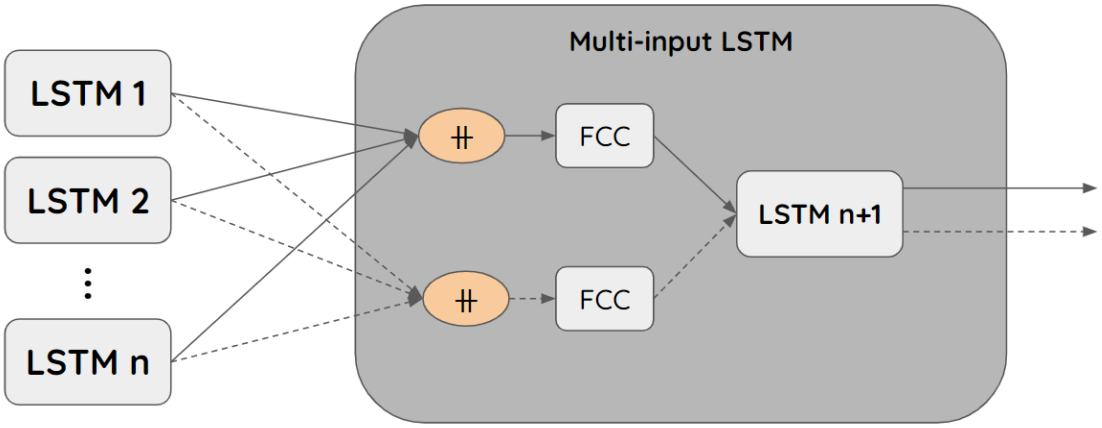
Figure 3 shows how to pass information from one LSTM cell to another. However, as shown in Figure 2c, it is possible to have multiple LSTM cells as inputs. At first, one could think that simply concatenating the different variables along their second axis could work. It is, of course, a possible solution. However, this would lead to increasing the size of the LSTM cells exponentially. Indeed, if the cell $\texttt{LSTM}_t$ has n_{anc} direct ancestors, the size of its inputs and outputs would then be of size $\mathcal{B} \times n\mathcal{H}$. For example, in Figure 5, the cell for the variable `work_status` would have a size of $2\mathcal{H}$ and, thus, the cell for `choice` would have a size of $12\mathcal{H}$ since 5 of its direct ancestors have the variable `work_status` has an ancestor. Thus, we have to find a way to keep the cell size consistent no matter how many direct ancestors there are. To achieve this, we propose two different solutions: the multi-input post-LSTM cell (Figure 4a) and the multi-input pre-LSTM cell (Figure 4b).

multi-input post-LSTM cell

This multi-input LSTM cell consists in using n_{anc} LSTM cells that will output the same variable. In the case of a continuous variables, each $\texttt{LSTM}_{n+1,l}$ would be composed of the two LSTM cells used to generate the continuous variables. We thus have a total of n_{anc} outputs $w_{t,l}$, cell states $C_{t,l}$, and attention vectors $a_{t,l}$. The outputs have to be transformed so that only one final variable is outputed in the model. The same applies to the cell states and attention vectors such that they can be passed to the next LSTM cell. We, therefore, concatenate each elements along the second axis, *e.g.* $C_{t,l}$, $l = 1, \dots, n$ are concatenated in a vector $\mathbf{C}_t \in \mathbb{R}^{\mathcal{B} \times n_{anc}\mathcal{H}}$. Finally, a FCC is used to transform $\mathbf{C}_t$ into



(a) multi-input post-LSTM cell



(b) multi-input pre-LSTM cell

Figure 4: Structure of the two multi-input LSTM cells. Figure (a) shows the fully connected layer after the LSTM cells and Figure (b) shows the fully connected layer before the LSTM cell.

$C_t \in \mathbb{R}^{\mathcal{B}\times\mathcal{H}}$. For the outputs, we use the same activation functions describe earlier. For the other variables, we do not use any activation function.

This way to reduce the dimensions of multiple outputs is valid. However, it has a couple of issues:

- The number of LSTM cells to be trained greatly increased. Indeed, we will have the same number of additional LSTM cells to be trained for each variable with multiple inputs. It thus leads to a larger training time.
- There are no interactions between the different inputs. Since we concatenate the results after the generation of the variable, each LSTM cell only sees one input. It is, thus, similar to taking a weighted average of the results instead of using more information.

multi-input pre-LSTM cell

This second version of the multi-input LSTM cell aims at fixing the issues with the first method. The idea is to concatenate the inputs of the cell $\texttt{LSTM}_{n+1}$ instead of the outputs. We, thus, have to concatenate the inputs f_t, the attention vectors a_t, and the states C_t of each direct ancestors of the cell $\texttt{LSTM}_{n+1}$. Each of these variables is concatenated on their second axis, similarly to the post-LSTM methodology. The FCC are all used to resize the variables and, thus, do not use any activation function. Therefore, the interaction between the different ancestors is directly taken into account while training each FCC. In addition, as seen in Figure 4b, this multi-input LSTM cell only uses one LSTM cell. Therefore, it does not change the complexity of the model by much.

3.2.2 LSTM sequence using a DAG

The graph representing the dependencies variables has to be an ensemble of Directed Acyclic Graph (DAG). A DAG is a graph with directed edges that does not contain any cycle. Therefore, it must follow the following rules:

- Each variable must be represented as a vertex in a graph. However, it is unnecessary to have edges between each vertex, *i.e.* some vertices or group of vertices can be separated.
- The graph cannot contain any cycles.

If these two conditions are met, the specified DAG can then be used to build the generator. However, one must use an algorithm to determine the order of the built variables in the generator. Algorithm 1 shows the algorithms used to order the variables before building the generator.

Algorithm 1 Ordering of the variables using a DAG

Inputs: DAG: $\mathcal{G}$
Output: ordered list of variables: $\mathcal{L}$

1: Compute a dictionary `in_edges` with the variable names as keys and the list of vertices that are the origin of an in-edge.
2: Initialize `untreated` as a set with all the variables names and `treated` an empty list
3: Initialize `to_treat` as a list containing all the variables with 0 in-edges
4: **while** $|$`untreated`$| > 0$ **do**
5: **for all** $n \in$ `to_treat` **do**
6: Remove n from `untreated` and add it to `treated`
7: Set `to_treat` as an empty list
8: **for all** $e \in \mathcal{G}.E$ **do** ▷ e is an edge and it is a tuple with 2 values: the out-vertex and the in-vertex
9: Initialize boolean `all_ancestors_treated` to True
10: **for all** $l \in$ `in_edges`$[e[1]]$ **do**
11: **if** $l \notin$ `treated` **then**
12: Set `all_ancestors_treated` to False
13: **if** $e[0] \in$ `treated` `AND` `all_ancestors_treated` is True `AND` $e[1] \notin$ `treated` `AND` $e[1] \notin$ `to_treat` **then**
14: Add $e[1]$ to the list `to_treat`
15: **return** `treated`

We start Algorithm 1 by selecting all the variables with 0 in-edges. We can choose these variables as the first ones since they do not have any ancestors. Then, we go through each edge of the DAG $\mathcal{G}$ on line 8. On lines 9-12, we make sure that all the ancestors of the given edge e have been selected first. Indeed, it is not possible to have a descendant appear before an ancestor in the list. Thus, the variable `all_ancestors_treated` is used to verify this condition. Finally, we can add the in-vertex of edge e in the list of vertices that have to be treated if:

- the out-vertex of e has been treated
- all the ancestors of e have been treated
- the in-vertex has not already been treated (to avoid duplicates)
- the in-vertex is not already part of the vertices that have to be treated (to avoid duplicates)

Once we have the correct order of variables, we can build the generator by looping on the variables in the list `treated`. However, three different cases exist depending on the number of in-edges of the current variable v:

- **v has 0 in-edges:** We use the usual single-input LSTM cell. However, we have to use the "zero" vectors as inputs as shown for the cell $\texttt{LSTM}_1$ in Figure 3.
- **v has 1 in-edge:** We use the usual single-input LSTM cell. The different inputs of the LSTM cell have to be taken as shown for the cell $\texttt{LSTM}_2$ in Figure 3.
- **v has more than one in-edge:** We have to use a multi-input LSTM cell taking into account all the direct ancestors of variable v.

3.3 Discriminator and Loss function

The discriminator and the loss function are the same between the TGAN and the DATGAN. However, we recall them in this section.

3.3.1 Discriminator

The discriminator is fully connected neural network with l layers. We concatenate $\left\{v_{1,j}, \mathbf{u}_{1,j}, v_{2,j}, \mathbf{u}_{2,j}, \ldots, v_{n_c,j}, \mathbf{u}_{n_c,j}, \widetilde{\mathbf{d}}_{1,j}, \ldots, \widetilde{\mathbf{d}}_{n_d,j}\right\}$ together as the input. The internal layers are given by

$$f_1^{(D)} = \text{LeakyReLU}\left(\text{BN}\left(W_1^{(D)}\left(v_{1:n_c} + \mathbf{u}_{1:n_c} + \widetilde{\mathbf{d}}_{1:n_d}\right)\right)\right) \tag{6}$$

$$f_i^{(D)} = \text{LeakyReLU}\left(\text{BN}\left(W_i^{(D)}\left(f_{i-1}^{(D)} + \text{diversity}\left(f_{i-1}^{(D)}\right)\right)\right)\right), \quad \forall i = 2, \ldots, l \tag{7}$$

where $+$ is the concatenation operator, diversity$(\cdot)$ is the mini-batch discrimination vector presented by Salimans et al. (2016). Each dimension of the diversity vector corresponds to the total distance between one sample and all the other samples in the mini-batch. The distance is computed using a learned distance metric. BN$(\cdot)$ is the batch normalization, and LeakyReLU$(\cdot)$ is the leaky rectified linear unit activation function. The output of the discriminator is a scalar computed the following way:

$$o = W^{(D)}\left(f_l^{(D)} + \text{diversity}\left(f_l^{(D)}\right)\right) \tag{8}$$

3.3.2 Loss function

This model is trained using Adam optimizer (Kingma and Ba, 2014). To warm up the mode more efficiently, we optimize the KL divergence of the categorical variables and the cluster vector of the continuously variables at the same time. It is done by adding them to the loss function. In addition, it is known that adding the KL divergence term can also make the model more stable. Therefore, the loss of the generator is given by:

$$\mathcal{L}_G = -\mathbb{E}_{z\sim\mathcal{N}(0,1)}\left[\log D(G(z))\right] + \sum_{i=1}^{n_c} KL(\mathbf{u}_i', \mathbf{u}_i) + \sum_{i=1}^{n_d} KL(\widetilde{\mathbf{d}}_i', \widetilde{\mathbf{d}}_i) \tag{9}$$

where $\mathbf{u}_i'$ and $\widetilde{\mathbf{d}}_i'$ are generated data, and $\mathbf{u}_i$ and $\widetilde{\mathbf{d}}_i$ are real data. The discriminator, on the other hand, is optimized using the conventional cross-entropy loss:

$$\mathcal{L}_D = -\mathbb{E}_{v_{1:n_c}, \mathbf{u}_{1:n_c}, \widetilde{\mathbf{d}}_{1:n_d}\sim\mathbb{P}(\mathbf{T})}\left[\log D\left(v_{1:n_c}, \mathbf{u}_{1:n_c}, \widetilde{\mathbf{d}}_{1:n_d}\right)\right] + \mathbb{E}_{z\sim\mathcal{N}(0,1)}\left[\log D(G(z))\right] \tag{10}$$

where $v_{1:n_c}, \mathbf{u}_{1:n_c}, \widetilde{\mathbf{d}}_{1:n_d}$ corresponds to $\left\{v_{1,j}, \mathbf{u}_{1,j}, v_{2,j}, \mathbf{u}_{2,j}, \ldots, v_{n_c,j}, \mathbf{u}_{n_c,j}, \widetilde{\mathbf{d}}_{1,j}, \ldots, \widetilde{\mathbf{d}}_{n_d,j}\right\}$.

4 Case Study

Currently, we have only one dataset that has been used to assess the performance of the DATGAN. We will add other datasets in the future. Nevertheless, this topic is further discussed in Section 6. The dataset we are using for this study-travel tracker survey was carried out between January 2007 and February 2008 by the Chicago Metropolitan Agency for Planning (CMAP). This dataset contains 87'946 trips (rows) and has 15 variables. All the variables are represented in the DAG in Figure 5. Amongst these variables, three of them are considered continuous: `age`, `distance`, and `departure_time`. The rest of the variables are either binary or categorical.

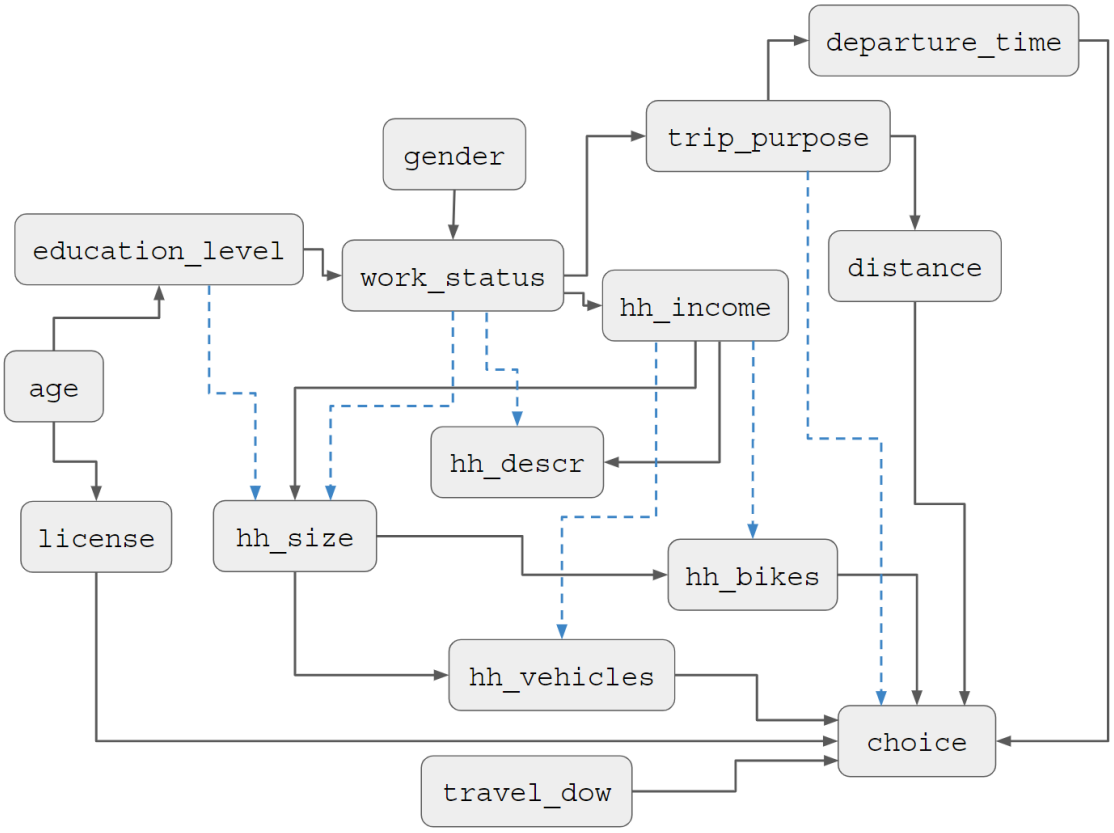


Figure 5: DAG representing the dependencies of the variables in the CMAP dataset. The dashed blue links are considered secondary links since a longer path exists between the two nodes. Therefore, the DAG with only the black arrows is the transitive reduction of the current graph.

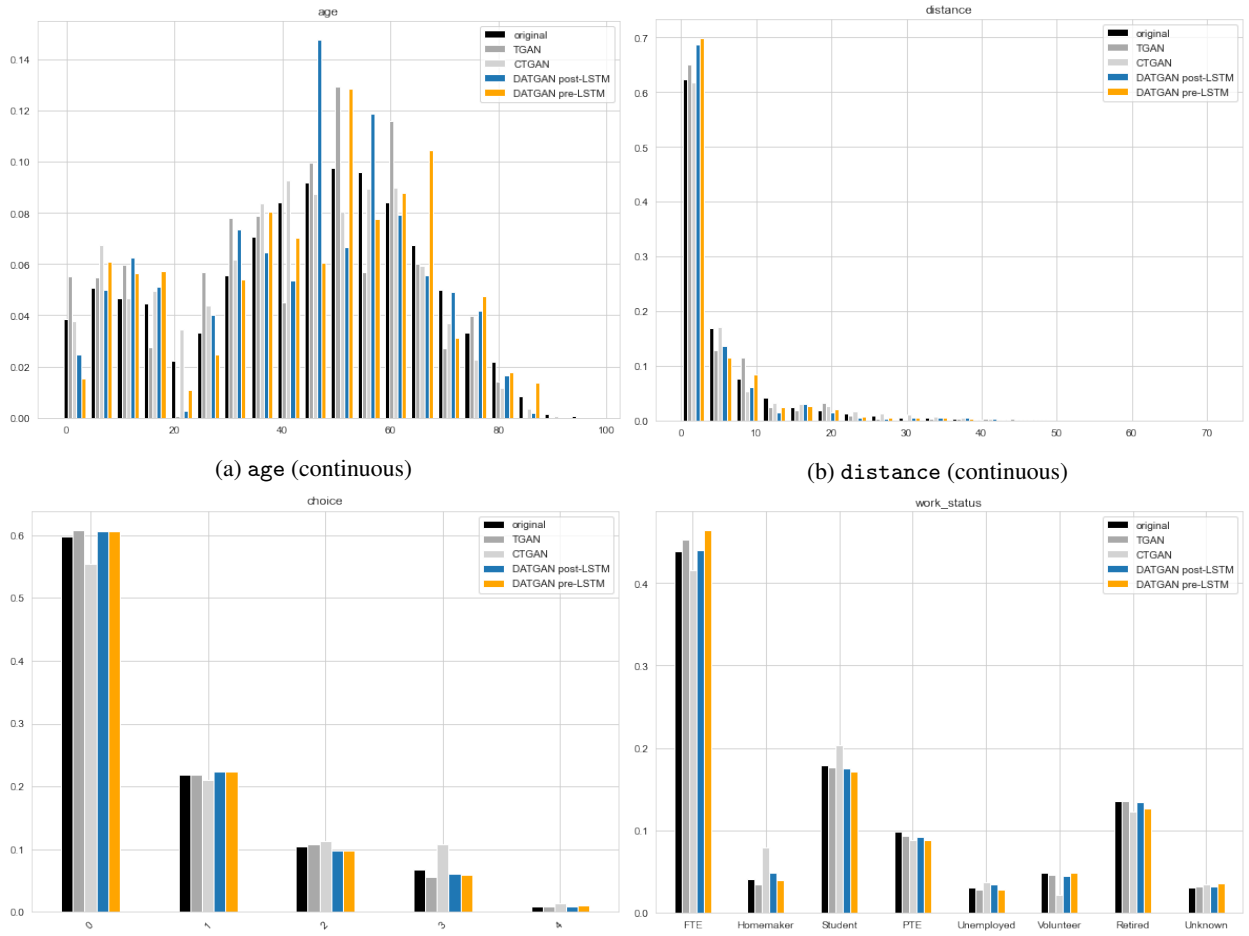
The DAG in Figure 5 has been designed around the idea of forecasting the variable `choice`. Indeed, we tried to create links such that the dependencies would explain the mode choice used in each trip. To analyse the sensitivity of the DATGAN on the DAG, we propose two versions of the DAG. The first one is the complete one with all the links presented in Figure 5. The second one is the transitive reduction of the DAG, in which we removed the blue dashed lines. This allows us to have a simpler graph and compare the results of the DATGAN with different DAG. In addition, both versions of multi-input LSTM cells are tested against two of the current state-of-the-art model for generating tabular data: Tabular GAN (TGAN) by Xu and Veeramachaneni (2018) and Conditional Tabular GAN (CTGAN) by Xu et al. (2019). All these models have been trained for 2'000 epochs.

5 Results

5.1 Comparison with state-of-the-art models

In this section, we present the results obtained by the two methods presented in the Methodology (Section 3) and compare them against some state-of-the-art methods. Since GANs do not have precise stopping criteria, we trained each model for 2000 epochs and generated a synthetic dataset using the resulting models. The first analysis we can make is on the distribution of each variable separately. Figure 6 shows some examples for four different variables.

Since these datasets contain multi-modal discrete data and continuous variables, we compute a frequency list for each variable. The maximum number of bins is set to 50. We can then compare the frequencies between the synthetic dataset and the original dataset as shown in Figure 7. These histograms allow us to draw two conclusions. The first one is that continuous variables are generally noisier than categorical variables. Indeed, the number of possible values is much higher, thus making the generating method less precise. In the case of TGAN and DATGAN, these continuous variables are generated using a GMM. CTGAN, on the other hand, is using a Variational Gaussian Mixture (VGM) model. The second conclusion is that CTGAN seems to be slightly worse than the other models for discrete variables



(a) `age` (continuous)

(b) `distance` (continuous)

(c) `choice` (categorical)

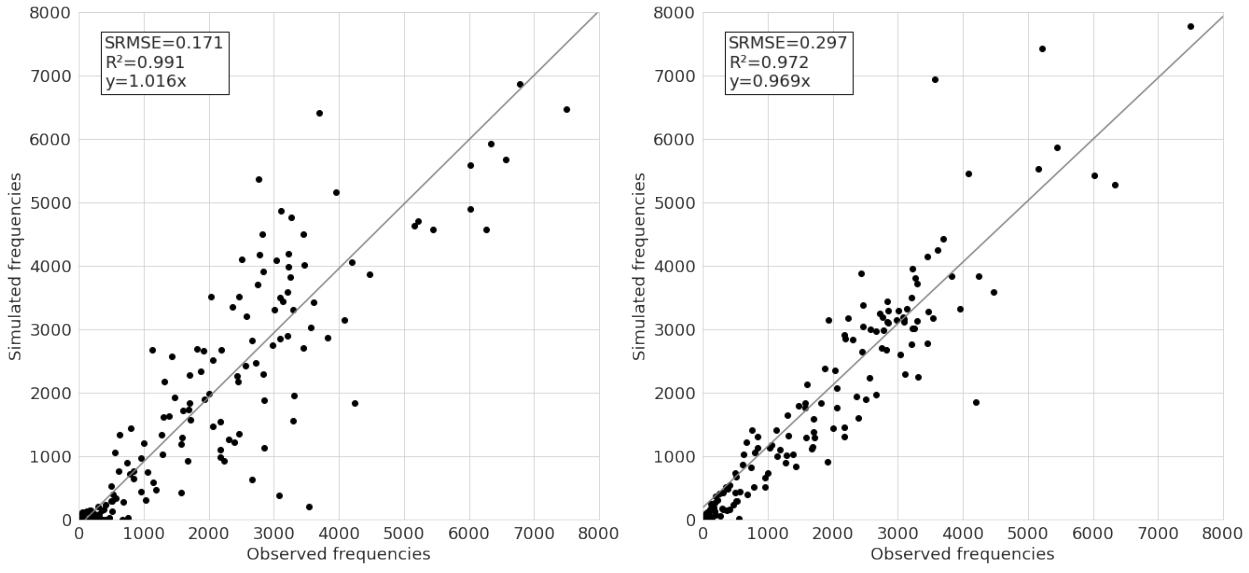
(d) `work_status` (categorical)

Figure 6: Qualitative comparison of some variables

with an equivalent training time. This is expected since all these models are one-hot encoding the categorical variables. However, the CTGAN is a more complex Neural Network than the TGAN or the DATGAN since it contains many more neurons.

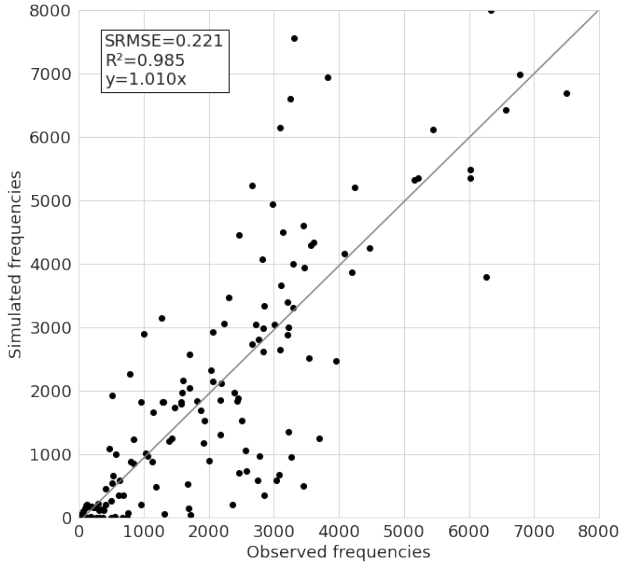
Visually comparing distributions is not sufficient to get a good assessment of the synthetic dataset. We, thus, follow the procedure given by (Müller and Axhausen, 2010) to compute the SRMSE on all the synthetic datasets. To compute the SRMSE, we first need to create frequencies for the generated data. For the categorical ones, we use the different categories as the bins. For the continuous variables, we create 50 bins and compute the frequencies on these bins. We can then compare the simulated frequencies against the observed ones and compute some statistics. Figure 7 shows the results for each model. In terms of SRMSE and R^2, the DATGAN pre-LSTM is the best model. In terms of slope, the DATGAN post-LSTM is slightly better. However, the slope is not the best assessment method since a larger dispersion of the data can lead to a better average, as seen in Figure 7c.

It is also possible to compute different statistics on each variable separately and study the results. For example, table 1 shows the average and standard deviation for five different statistics. THe DATGAN pre-LSTM has the best average for four of these statistics and the CTGAN for one. It is interesting to note that the DATGAN pre-LSTM shows consistently better results than the DATGAN post-LSTM. While the latter contains more LSTM cells, it seems that at an equivalent training time, fewer LSTM cells leads to a better final model. In addition, we see that CTGAN is the worst model if we use the RMSE and the MAE statistics. However, it is the most consistent one with the smallest standard deviation. This is explained by the fact that the TGAN and the DATGAN are less good at generating continuous variables than the CTGAN. For example, Table 2 shows the comparison of the average SRMSE on continuous and categorical variables. The CTGAN is consistent across both types of data. However, the other models are much better with discrete variables.

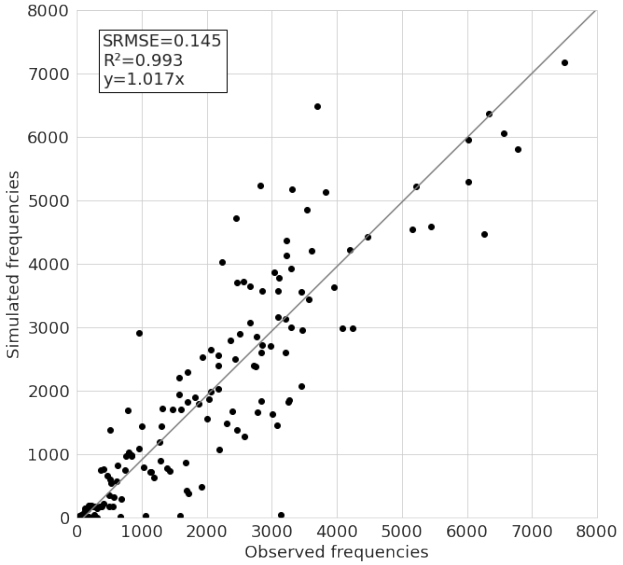


(a) TGAN

(b) CTGAN



(c) DATGAN post-LSTM



(d) DATGAN pre-LSTM

Figure 7: Frequency and SRMSE for the three models

Since the CMAP dataset is mainly comprised of categorical variables, the DATGAN models show better results across the different statistics.

	SRMSE	RMSE	MAE	R^2	Pearson
TGAN	0.161 ± 0.244	744.6 ± 343.3	555.9 ± 267.5	0.900 ± 0.234	0.970 ± 0.071
CTGAN	0.189 ± 0.106	1947.5 ± 1244.7	1447.5 ± 840.5	0.915 ± 0.088	0.962 ± 0.042
DATGAN post-LSTM	0.212 ± 0.364	776.5 ± 511.0	557.4 ± 270.5	0.819 ± 0.444	0.953 ± 0.113
DATGAN pre-LSTM	0.147 ± 0.220	661.8 ± 273.6	498.9 ± 197.0	0.906 ± 0.222	0.971 ± 0.071

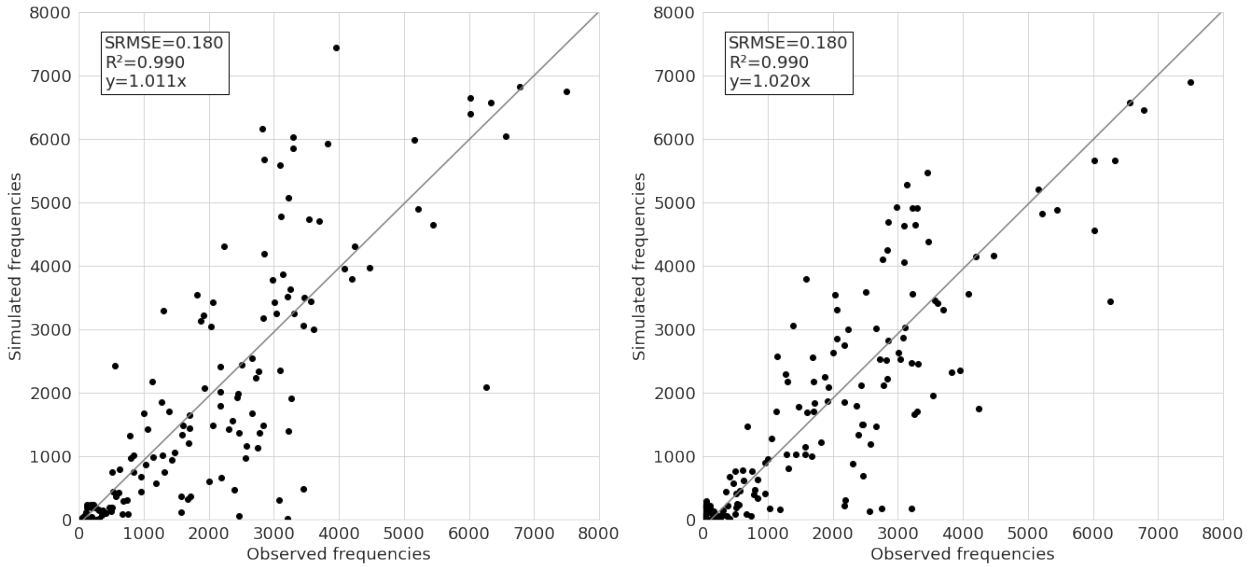
Table 1: Statistics for the different models (average over all variables)

	Continuous	Categorical
TGAN	0.631 ± 0.146	0.044 ± 0.021
CTGAN	0.221 ± 0.027	0.181 ± 0.117
DATGAN post-LSTM	0.899 ± 0.265	0.040 ± 0.020
DATGAN pre-LSTM	0.578 ± 0.084	0.039 ± 0.019

Table 2: Comparison of the SRMSE values for the continuous and the categorical variables

5.2 Sensitivity analysis on the DAG

We have shown that the DATGAN with a complete DAG such as the one given in Figure 5 leads to a more accurate synthetic dataset. However, we want to know the impact of the DAG on the results. It would not make sense to generate a DAG that is only comprised of nodes. In addition, the TGAN corresponds to a DATGAN with only one edge exiting and entering each node. We can, thus, already conclude that a complete DAG leads to better results. However, it is interesting to investigate the effect of the transitive reduction on the DAG in Figure 5. Figure 8 shows the same results as Figure 7 for the two DATGAN models with the reduced DAG. Interestingly, the DATGAN post-LSTM has better results with this simplified graph, while the DATGAN pre-LSTM shows worse results. Table **??** confirms these results with the other statistics. These results are most likely due to the fact that removing some of these edges made the DATGAN post-LSTM a lot simpler and easier to train, thus achieving better training status. However, none of these models can generate better results than the DATGAN pre-LSTM with the complete DAG. We, thus, recommend the user to create a graph as complete as possible and use the DATGAN pre-LSTM.



(a) DATGAN post-LSTM

(b) DATGAN pre-LSTM

Figure 8: Frequency and SRMSE for the two DATGAN with transitive reduction

	SRMSE	RMSE	MAE	R^2	Pearson
DATGAN post-LSTM	0.212 ± 0.364	776.5 ± 511.0	557.4 ± 270.5	0.819 ± 0.444	0.953 ± 0.113
DATGAN post-LSTM (trans. red.)	0.175 ± 0.273	732.3 ± 357.8	553.2 ± 273.2	0.845 ± 0.355	0.958 ± 0.100
DATGAN pre-LSTM	0.147 ± 0.220	661.8 ± 273.6	498.9 ± 197.0	0.906 ± 0.222	0.971 ± 0.071
DATGAN pre-LSTM (trans. red.)	0.174 ± 0.236	876.0 ± 406.6	654.3 ± 285.8	0.857 ± 0.332	0.960 ± 0.096

Table 3: Statistics for the different models (average over all variables)

6 Conclusion

In this article, we presented a new deep learning model based on GANs to generate synthetic populations. This model aims to add expert knowledge to deep learning models such that humans can better control how synthetic populations are generated. This is done by creating a DAG to represent the relationship between the different variables in a dataset. This DAG is then used as the structure for the LSTM cells. We presented two versions of the DATGAN: the pre-LSTM DATGAN and the post-LSTM DATGAN. These models have been tested against multiple state-of-the-art deep learning models to generate tabular data: TGAN and CTGAN. Preliminary results show that the pre-LSTM DATGAN outperforms the other methods using multiple statistics: SRMSE, RMSE and MAE. The results are similar to TGAN using R^2 or Pearson correlation. A short analysis of the complexity of the DAG has also been conducted. Early results suggest that a more complex DAG systematically leads to more accurate results.

In the future, we will extend our work in multiple ways. The most obvious one is to add more datasets in the comparison of these models. The next dataset that we will use is the London Passenger Mode Choice (LPMC) dataset by Hillel et al. (2018). While the topic is similar, this dataset is more complex than the CMAP dataset. We will also continue to work on the assessment of the results. While the SRMSE has been used in many articles, we believe it is not the most accurate assessment for synthetic populations. We, thus, plan to work on finding more accurate assessment methods for synthetic population generation. Additionally, we would like to extend the sensitivity analysis on the DAG. As the preliminary results suggest, the complexity of the graph influences the results. However, we do not have a good representation of the effect yet. We will thus continue to investigate this. Finally, as seen in the results, the method for continuous variables of the CTGAN is much better than the TGAN and DATGANs. Indeed, the CTGAN uses a Variational Gaussian Mixture (VGM) model, while the other models use a standard Gaussian Mixture Model. Thus, we would like to replace the GMM of the DATGAN with a VGM to see if we can further improve our current results.

References

Alqahtani, H., Kavakli-Thorne, M., and Kumar, G. (2021). Applications of Generative Adversarial Networks (GANs): An Updated Review. *Archives of Computational Methods in Engineering*, 28(2):525–552.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein GAN. *arXiv:1701.07875 [cs, stat]*. arXiv: 1701.07875.

Auld, J. A., Mohammadian, A. K., and Wies, K. (2009). Population Synthesis with Subregion-Level Control Variable Aggregation. *Journal of Transportation Engineering*, 135(9):632–639. Number: 9 Publisher: American Society of Civil Engineers.

Badu-Marfo, G., Farooq, B., and Paterson, Z. (2020). Composite Travel Generative Adversarial Networks for Tabular and Sequential Population Synthesis. *arXiv:2004.06838 [cs, stat]*. arXiv: 2004.06838.

Barbedo, J. G. A. (2018). Impact of dataset size and variety on the effectiveness of deep learning and transfer learning for plant disease classification. *Computers and Electronics in Agriculture*, 153:46–53.

Barthelemy, J. and Toint, P. L. (2013). Synthetic Population Generation Without a Sample. *Transportation Science*, 47(2):266–279. Publisher: INFORMS.

Beckman, R. J., Baggerly, K. A., and McKay, M. D. (1996). Creating synthetic baseline populations. *Transportation Research Part A: Policy and Practice*, 30(6):415–429. Number: 6.

Bhat, C. and Koppelman, F. (2003). Activity-Based Modeling of Travel Demand. volume 56, pages 39–65.

Borysov, S. S., Rich, J., and Pereira, F. C. (2019). How to generate micro-agents? A deep generative modeling approach to population synthesis. *Transportation Research Part C: Emerging Technologies*, 106:73–97.

Casati, D., Müller, K., Fourie, P. J., Erath, A., and Axhausen, K. W. (2015). Synthetic Population Generation by Combining a Hierarchical, Simulation-Based Approach with Reweighting by Generalized Raking. *Transportation Research Record: Journal of the Transportation Research Board*, (2493). ISBN: 9780309369633 Number: 2493.

Deming, W. E. and Stephan, F. F. (1940). On a Least Squares Adjustment of a Sampled Frequency Table When the Expected Marginal Totals are Known. *Annals of Mathematical Statistics*, 11(4):427–444. Number: 4 Publisher: Institute of Mathematical Statistics.

Farooq, B., Bierlaire, M., Hurtubia, R., and Flötteröd, G. (2013). Simulation based population synthesis. *Transportation Research Part B: Methodological*, 58:243–263.

Frégier, Y. and Gouray, J.-B. (2020). Mind2Mind : transfer learning for GANs. *arXiv:1906.11613 [cs, stat]*. arXiv: 1906.11613.

Garrido, S., Borysov, S. S., Pereira, F. C., and Rich, J. (2019). Prediction of rare feature combinations in population synthesis: Application of deep generative modelling. *arXiv:1909.07689 [cs, stat]*. arXiv: 1909.07689.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Networks. *arXiv:1406.2661 [cs, stat]*. arXiv: 1406.2661.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. (2017). Improved Training of Wasserstein GANs. *arXiv:1704.00028 [cs, stat]*. arXiv: 1704.00028.

Hillel, T., Elshafie, M. Z. E. B., and Jin, Y. (2018). Recreating passenger mode choice-sets for transport simulation: A case study of London, UK. *Proceedings of the Institution of Civil Engineers - Smart Infrastructure and Construction*, 171(1):29–42. Number: 1.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780. Conference Name: Neural Computation.

Jeon, H., Bang, Y., Kim, J., and Woo, S. S. (2020). T-GD: Transferable GAN-generated Images Detection Framework. *arXiv:2008.04115 [cs]*. arXiv: 2008.04115.

Jha, A., Chandrasekaran, A., Kim, C., and Ramprasad, R. (2019). Impact of dataset uncertainties on machine learning model predictions: the example of polymer glass transition temperatures. *Modelling and Simulation in Materials Science and Engineering*, 27(2):024002. Publisher: IOP Publishing.

Kim, J. and Lee, S. (2016). A simulated annealing algorithm for the creation of synthetic population in activity-based travel demand model. *KSCE Journal of Civil Engineering*, 20(6):2513–2523. Number: 6.

Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*. arXiv: 1412.6980.

Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*. arXiv: 1312.6114.

Linjordet, T. and Balog, K. (2019). Impact of Training Dataset Size on Neural Answer Selection Models. In Azzopardi, L., Stein, B., Fuhr, N., Mayr, P., Hauff, C., and Hiemstra, D., editors, *Advances in Information Retrieval*, Lecture Notes in Computer Science, pages 828–835, Cham. Springer International Publishing.

Liu, M.-Y. and Tuzel, O. (2016). Coupled Generative Adversarial Networks. *arXiv:1606.07536 [cs]*. arXiv: 1606.07536.

Liu, Y., Peng, J., Yu, J. J. Q., and Wu, Y. (2019). PPGAN: Privacy-preserving Generative Adversarial Network. *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 985–989. arXiv: 1910.02007.

Miranda, D. (2019). Reviewing Synthetic Population Generation for Transportation Models Over the Decades.

Mirza, M. and Osindero, S. (2014). Conditional Generative Adversarial Nets. *arXiv:1411.1784 [cs, stat]*. arXiv: 1411.1784.

Müller, K. and Axhausen, K. (2010). Population synthesis for microsimulation: State of the art.

Noguchi, A. and Harada, T. (2019). Image Generation From Small Datasets via Batch Statistics Adaptation. *arXiv:1904.01774 [cs]*. arXiv: 1904.01774.

Park, N., Mohammadi, M., Gorde, K., Jajodia, S., Park, H., and Kim, Y. (2018). Data Synthesis based on Generative Adversarial Networks. *Proceedings of the VLDB Endowment*, 11(10):1071–1083. Number: 10 arXiv: 1806.03384.

Philips, I., Clarke, G., and Watling, D. (2017). A Fine Grained Hybrid Spatial Microsimulation Technique for Generating Detailed Synthetic Individuals from Multiple Data Sources: An Application To Walking And Cycling. *International Journal of Microsimulation*, 10(1):167–200. Number: 1 Publisher: International Microsimulation Association.

Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv:1511.06434 [cs]*. arXiv: 1511.06434.

Rich, J. (2018). Large-scale spatial population synthesis for Denmark. *European Transport Research Review*, 10(2):63. Number: 2.

Rubin, D. B. (1973). The Use of Matched Sampling and Regression Adjustment to Remove Bias in Observational Studies. *Biometrics*, 29(1):185–203. Publisher: [Wiley, International Biometric Society].

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved Techniques for Training GANs. *arXiv:1606.03498 [cs]*. arXiv: 1606.03498.

Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6(1):60.

Wang, Y., Gonzalez-Garcia, A., Berga, D., Herranz, L., Khan, F. S., and van de Weijer, J. (2020). MineGAN: effective knowledge transfer from GANs to target domains with few images. *arXiv:1912.05270 [cs]*. arXiv: 1912.05270.

Xu, L., Skoularidou, M., Cuesta-Infante, A., and Veeramachaneni, K. (2019). Modeling Tabular data using Conditional GAN. In Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F. d., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 7335–7345. Curran Associates, Inc.

Xu, L. and Veeramachaneni, K. (2018). Synthesizing Tabular Data using Generative Adversarial Networks. *arXiv:1811.11264 [cs, stat]*. arXiv: 1811.11264.

Yin, D. and Yang, Q. (2018). GANs Based Density Distribution Privacy-Preservation on Mobility Data. *Security and Communication Networks*, 2018:e9203076. Publisher: Hindawi.