Generative adversarial networks for network traffic feature generation

Tertsegha J. Anande, Sami Al-Saadi & Mark S. Leeson

To cite this article: Tertsegha J. Anande, Sami Al-Saadi & Mark S. Leeson (2023) Generative adversarial networks for network traffic feature generation, International Journal of Computers and Applications, 45:4, 297-305, DOI: 10.1080/1206212X.2023.2191072

To link to this article: https://doi.org/10.1080/1206212X.2023.2191072

© 2023 The Author(s). Published by Informa UK Limited, trading as Taylor & Francis Group

Published online: 28 Mar 2023.

Article views: 1306

View related articles

View Crossmark data
Generative adversarial networks for network traffic feature generation

Tertsegha J. Anande, Sami Al-Saadi and Mark S. Leeson
School of Engineering, University of Warwick, Coventry, UK

ABSTRACT

Generative Adversarial Networks (GANs) have remained an active area of research, particularly due to their increased and advanced evolving application capabilities. In several domains such as images, facial synthesis, character generation, language processing and multimedia, they have been implemented for advanced tasks. However, there has been more limited progress in network traffic data generation due to the complexities associated with data formats and distributions. This research implements two GAN architectures that include data transforms to simultaneously train and generate categorical and continuous network traffic features. These architectures demonstrate superior performance to the original ‘Vanilla’ GAN approach, which is included as a baseline comparator. Close matches are obtained between logarithms of the means and standard deviations of the fake data and the corresponding quantities from the real data. Moreover, similar principal components are exhibited by the fake and real data streams. Furthermore, some 85% of the features from the fake data could replace those in the real data without detection.

1. Introduction

Generative Adversarial Networks (GANs) utilize two models, the Generator and the Discriminator, which compete to prove that input data are fake or real and are trained using signal backpropagation. GANs were first introduced by Goodfellow et al. [1] in an architectural form that has become known as the Vanilla GAN, to which several enhancements have been made in recent years [2]. They have been the subject of increased attention due to their versatility and dynamic applicability. In particular, GANs are generally popular for their applications in image synthesis and image related tasks where the models are trained to reproduce images, transform images in one form to another or translate other data formats to images [3]. GAN implementations for generation and synthesis in the image domain are thus at advanced stages as shown in [4] and several others, with several improved successes on previous research works. Human faces, medical, audio, video, text, anime and handwritten characters, and language are other domains actively under investigation for the application of GANs [5–7].

Developments such as the Packet Generation of Network Traffic GAN (PAC-GAN) [8], discussed in Section 2 below, have led to the successful learning of intricate data distribution features and the generation of synthetic data that are hard to distinguish from a real data distribution. With more and evolving applications, this hybrid structured deep, generally unsupervised learning model has become a popular method for synthetic data generation in areas such as finance [9] and population data [10].

GANs have only very recently been successfully implemented in network traffic data generation but the area is relatively undeveloped. Compared to other domains, network traffic data generation and synthesis using GANs does not enjoy the same modeling flexibility due to data imbalance, mixed data types, and multimodal and non-Gaussian data distributions [11]. Hence, reproducing realistic synthetic network traffic data is a challenging task.

In this study, we access GANs' general application capabilities particularly with respect to the network traffic data generation domain. We further make key contributions that include:

(1) Implementing two GAN architectures suitably modeled to incorporate the reversible data transformation hyper transformer, and Gaussian copula transformer to generate synthetic network traffic data features.

(2) Reproducing realistic categorical and continuous synthetic network traffic features that show GANs performance in the network traffic data generation domain.

(3) Demonstrating that features from the synthetic data can successfully pass as real data features without detection.

The paper is divided as follows. Section 2 explores various GAN models implemented for network traffic generation or related tasks and outlines the contributions of this work. Section 3 introduces the basic GAN architectures and the improved models implemented in this study. Details of the simulation experiment including data collection, pre-processing, transformation, and training are presented in Section 4, with further discussions and evaluation of the results in Section 5. Finally, Section 6 completes the paper with conclusions and observations concerning future prospects.

2. Related work

The initial GAN architecture [1] exhibited certain shortfalls including the inability of the Generator model (G) to sufficiently learn all data distribution modes. This meant that it would fail to improve on its generation after a few training steps and produce results in the constant production of just one mode, known as mode collapse [2]. To overcome these limitations, several improvements were made to the Vanilla GAN framework including adding class conditions.
to enhance data generation representations (Conditional GANs) [12], incorporating convolutional layers for enhanced image generation and regeneration (Deep Convolutional GANs) [13], replacing and modifying the loss functions (Wasserstein GANs) [14], adding inference network extensions, and adversarial training for enhanced robustness and model training convergence speed [15]. Thus, GANs have continued to gain increased attention due to their versatility and dynamic applicability, with increased and evolving applications too numerous to fully document here (see the review by Alqahtani et al. [7]) and including network traffic data, which is the focus of this study.

Cheng [8] modified the Conditional GAN framework to include inverse and conventional Convolutional Neural Network (CNN) architectures to produce the PAC-GAN model. This was implemented to encode and convert individual network traffic packet byte values, represent, and duplicate the converted values for one-to-multi mapping, and further generate network traffic packets for transmission through the internet [8]. The implementation recorded a success rate of 87.7%, averaged over generating, manipulating, and eliciting responses for Internet Control Message Protocol (ICMP) Pings, Domain Name Server (DNS) queries and Hyper Text Transfer Protocol (HTTP) get requests at the Internet Protocol (IP) individual packet byte level [8]. Although this was the first successful attempt at generating network traffic data above metadata/flow-level, it was not possible to generate multi-serial packets from varied network traffic data types. In this approach, network traffic data were first converted into 28 x 28 square matrices and encoded before being fed as input into the PAC-GAN model for training.

Previously, Ring et al. [16] used the Wasserstein GAN with Gradient Penalty (WGAN-GP) model with an incorporated Two Time-Scale Update Rule (TTUR) to generate and transform flow-based network traffic. Their approach, Flow-Based Network Traffic Generation GAN, converted categorical features to binary attributes while learning vector representations using the IP2Vec similarity measure [16]. To test the WGAN-GP, three methods to investigate synthetic feature generation were employed. In the first, IP addresses and Port numbers were transformed to categorical features, while other categorical features were normalized to the interval [0,1], which unfortunately produced unwanted similarities and significant errors between them. The second method mapped each octet of an IP address to an 8-bit binary representation producing a 32-bit representation, Port numbers were transformed to a 16-bit representation, bytes and packets to a limited length representation of 32-bits, and duration normalized to [0,1]. The approach successfully generated flow-based network traffic including previously unseen IP addresses and port numbers. The third method embedded categorical features into an m-dimensional continuous feature space for input, generated flows mapped to embeddings and re-transformed to their original space, thus surpassing the other two approaches in performance with an average evaluation of 99.83%. This model only accepted numerical features for training as all categorical and non-numerical features were transformed or embedded during data pre-processing and prior to training.

Dowoo et al. [17] proposed the Packet Capture File Generator Style-based GAN (PcapGAN) to generate, augment and analyse network traffic data packet captured from real traffic flows. Packet data (Pcap) features were extracted and converted to graph (source and destination IPs), image (time interval) and layer sequences, which were fed into the model. There they were converted into combinations of protocols and the augmented sequence of identical numbers (option data) used to create packet data for each protocol. The time interval information was used to randomly set the first packet’s start time, the reception time of other packets and to sort the generated packets at each host; these were then transformed into a Pcap file [17]. In distinguishing between the original data and synthetic data, the similarity score from the evaluation was 0.5 on a scale of 0–1 (0 being completely dissimilar and 1 being indistinguishable). Pcap-GAN had the disadvantage that it could only train with data that were converted to graph, image and layer sequences prior to training.

A unidirectional and Recurrent Neural Network (RNN) was modeled using a Long Short-Term Memory (LSTM) architecture and incorporated on the GAN framework to imitate network traffic flows of Facebook chats for malicious purposes while mimicking flows of legitimate users [18]. The Facebook chat network traffic GAN used the HTTP web service to communicate and expose API calls, get parameters and send feedback to the malware, while facilitating channel communication to the server and the Internet. Network traffic flow features (including network flow duration, total number of bytes in flow, calculated inter-flow time from timestamp for each flow) were converted into time series prior to training with the model, meaning it was only able to train with continuous features.

The models reviewed above required categorical and non-numeric data features to be encoded or converted, and then transformed before being suitable for use in model training. In contrast, this research implements methods based on improved GAN architectures that can train without any such conversion prior to feeding into the models for training. Here, we consider training with unconverted and converted categorical data features, and show the limitations associated with the latter before feeding into the network. We reproduce realistic categorical and continuous synthetic network traffic features to demonstrate the performance of appropriate GANs in the network traffic data generation domain. We also show that features from our synthetic data can successfully pass as real data features without detection. First, we establish the parameters and characteristics of the GANs considered. Hence, the next section discusses the Vanilla GAN architecture, to establish a baseline for comparison, followed by the two improved GAN architectures implemented in this study.

3. Methodology

This section discusses the baseline GAN and two GAN architectures implemented in this study for generating synthetic network traffic data.

3.1. Vanilla GAN

The Vanilla GAN architecture incorporates two models in a corresponding minimax two-player game framework [1]. We describe in outline the training process and many of the characteristics of this are carried forward into the training of the other GAN models. The Generator model (G) takes in n random noise samples \(z_1, z_2, z_3, ..., z_{n-1}\) from a distribution \(P_z(z)\), then transforms these into a data vector \(G(z; \theta_g)\) that is presented to the Discriminator model (D) as a potential data sample. \(G\) and \(D\) are differentiable functions comprising artificial neural networks with parameters \(\theta_d\) and \(\theta_g\) respectively. The input to \(D\) is the generated data instance \(G(z; \theta_g)\) and \(n\) random samples \(x_1, x_2, x_3, ..., x_{n-1}\) of real data \(P_{data}(x)\), and produces an output \(D(x; \theta_d)\) with a score that shows the likelihood of \(G(z; \theta_g)\) being from the original data distribution. The GAN is trained over number of iterations, or epochs, and generally employs the Adam optimizer [19] to update the parameters until satisfactory performance (however defined) is achieved.

To confuse \(G\) and \(D\) attempts to make both \(G(z)\) and \(P_{data}(x)\) equal to 0.5. \(D\), on the other hand, aims to force \(D(x) = 1\), thereby correctly distinguishing the real data from the synthetic data. A major setback with the Vanilla GAN model remains the inability of \(G\) to sufficiently
learn all data distribution modes thereby failing to improve on its generation after a few training steps and resulting in mode collapse.

3.2. Conditional tabular GAN (CTGAN)

CTGAN is based on the Conditional GAN (CGAN) architecture that combines extra data from class labels (y) with input for G and D [12] and uses the Wasserstein GAN (WGAN) architecture. This (a) replaces both D with a Critic model (C); (b) utilizes the Wasserstein loss (W-Loss) based on the Earth Mover Distance (EMD) for approximating the distance between \( P_{\text{data}}(x) \) and \( G(z; \theta_D) \) [20] rather than the Jensen Shannon distance metric based Binary Cross-Entropy (BCE) loss function; (c) includes a penalty parameter (\( \lambda \)) [21]. The architecture is further improved to incorporate a Conditional G, and uses the sampling training approach that resamples training data and includes all discrete variable categories in the learning samples [21]. Both G and C have fully connected hidden layers that capture relations between features, mixed activation functions to handle non-linear interactions between features, and dropout techniques that prevent overfitting at the same time addressing imbalance problems that affect discrete features [11].

To ensure that multimodal distributions of continuous features are properly represented, CTGAN first uses the variational Gaussian mixture model (VGM) [22] to estimate the number of modes in a column and fit a Gaussian mixture to the continuous features. Then, it implements the Mode Specific Normalization (MSN) technique that, as illustrated by Figure 1 [11], computes the probability of each mode for each value, selects and normalizes the mode sample. The process is completed with a one-hot vector with a normalized scalar in the appropriate position for the mode to indicate its value.

3.3. Copula GAN

Given two independent random variables, \( X \) and \( Y \), the marginal cumulative density functions (CDFs), \( F_X \) and \( F_Y \) describe how each variable is distributed, and the joint CDF \( F_{XY} \) provides their joint distribution. These may be related by \( F_{XY} = C(F_X, F_Y) \), where \( C \) is a known as a copula and has been used by Kamthe et al. [23] to produce a variant of the CTGAN architecture, Copula GAN. This combines the copula approach, using probability integral transforms to establish relationships between uniform marginals and the GAN framework to implement a probabilistic G model while estimating univariate marginals and the multivariate copula density in the learning process. The data features are transformed using the distribution that best achieves uniformity and then the relationship between the transformed features is modeled. Copula GAN applies the Gaussian copula CDF based transformation rather than VGM to enhance the learning process. The G model implements inverse transformation sampling to randomly generate samples from a uniform distribution.

Kamthe et al. [23] implemented a copula flow model to learn three marginal densities of mixed variables using three bivariate copulas (Gaussian, Clayton, and Gumbel). The Gaussian, Clayton and Gumbel copulas were represented thus [24]:

\[
\begin{align*}
    \text{Gaussian: } & C^\text{G}(u, v; \theta) = \Phi_G(\Phi^{-1}(u), \Phi^{-1}(v); \theta), -1 < \theta < 1 \\
    \text{Clayton: } & C^\text{C}(u, v; \theta) = (u^{-\theta} + v^{-\theta} - 1)^{-1/\theta}, \theta > 0 \\
    \text{Gumbel: } & C^\text{G}(u, v; \theta) = \exp[-\{(\ln u)^\theta + (\ln v)^\theta\}^{1/\theta}], \theta \geq 1
\end{align*}
\]

where \( \Phi \) was the CDF of the standard normal distribution and \( \Phi_G(\cdot, \cdot; \theta) \) was the standard bivariate normal distribution and \( \theta \) was the degree of correlation of the variables. The Clayton Copula captured variable distribution changes at lower tail dependencies while the Gumbel Copula captured similar changes of in the upper tail dependencies. Thus, the Copula GAN model was able to successfully learn discrete and continuous features.

4. Simulation

4.1. Data collection

To investigate the performance of the GAN types listed above we used the UNSW_NB15 dataset [25], which is a captured record of contemporary network traffic flows comprising both normal and malicious activities. This dataset was not used as an absolute representation of network traffic data in all situations but as an illustration to show the capability of GANs in the network traffic data generation domain. It contained 49 features, including Source and Destination Internet Protocol (IP) addresses and Port numbers, protocol types, duration of flows, transmitted bytes, and packets for both Source and Destination, normal and malicious flow types, attack categories, and 35 other traffic flow statistics [26]. The set had 56,000 normal flows, and 164,672 malicious flows including generic malware, exploits, Denial of Service (DoS), reconnaissance, backdoor, shellcode and worms.

4.2. Data pre-processing

Originally stored in four CSV files, all files were concatenated into a single dataset containing 2,540,048 records. There were nine (9) categorical features and forty (40) continuous features. The label feature
in the original set contained various attack categories and sometimes
not a number (NaN) values instead of normal features because of
missing or corrupted data in the UNSW_NB15 dataset. Such NaN
values were replaced with normal traffic features and the dataset nor-
malized. A portion of the set (1000 rows and 65,230 records extracted
from all features) was used for training to overcome physical memory
limitations when trying to perform one hot encoding on the entire
dataset. All dataset features were included in the training and testing
sets. For training, 70% of the dataset was used while 30% was used for
testing.

4.3. Data transformation

In this section, we discuss the data transformation methods used in
the GAN models employed.

4.3.1. Vanilla GAN

Machine learning algorithms often perform best when the input data
have a Gaussian distribution [27]. However, this is not always the
case and so transforming the data to have a more Gaussian distribu-
tion by using power transforms that employ power functions such
as exponents to achieve this. Thus, one of these (the Yeo-Johnson
power transform [28]) was used on all the training set fields to pro-
duce a Gaussian distribution, eliminate skewness and stabilize data
variance. The method also allowed for zero and negative values in
the set. An inverse transformation was carried out on the synthetic
data to enable evaluation and visualization of the results generated.

4.3.2. CTGAN

For this GAN, we made use of the Reversible Data Transforms
(RDT) library [29], which provides functions for (i) cleaning data
and preparing them for modeling techniques; (ii) transforming the
output back into its original format. Within the library, the hyper
transformer employs a set of transformer methods that convert non-
numeric data input into a numerical form since many modeling
approaches require only numerical inputs. Thus, this was used to
transform all categorical inputs to numbers for training, after which
the hyper transformer’s reverse transform method was used to return
all variables to their original formats.

4.3.3. Copula GAN

In this case, the Gaussian Copula Transformer was used that applied
the CDF based transformation on both continuous and categorical
features and used RDT to ensure that the dataset features were trans-
formed to a format suitable for training with the synthesizer. RDT
transformed all categorical features to numeric representations to
enable the model to learn each feature’s probability distribution, con-
verted them to marginal distribution CDF values, and then obtained
their standard norm using the inverse CDF transformation. This
process was inverted after Copula GAN training by first comput-
ing the standard normal CDF, applying inverse CDF of the features’
marginal distributions, and the RDT transformations that returned
the categorical features to their original format.

4.4. Training

Parameters: Vanilla GAN training was set to run for 3000 epochs to
observe and allow for ample learning time. Initial results indicated
that the GAN stopped learning after 1880 epochs when mode col-
lapse occurred although training was progressed beyond this point
to ensure that this was indeed the case rather than merely difficulty
in learning. After several training runs, the CTGAN and Copula
GAN models were set to train for 1000 epochs as results showed
that they exhibited better generation properties than the Vanilla
GAN – the number of epochs was limited to 1000 due to restricted
system resources vis-à-vis training time (14:45h – CTGAN and
14:20h – Copula GAN). All models were optimized using Adam
optimization, the Vanilla GAN learning rate was set at $\frac{5}{10^{0.3}}$,
while both CTGAN and Copula GAN had learning rates for $G$ and
C of $2 \times 10^{-4}$.

System Configuration: Training was performed on a 3 GHz Core
i7 (8 Cores) CPU with 32GB RAM running Windows 10 and imple-
mented in JupyterLab on the Anaconda Navigator platform. Vanilla
GAN employed Python 3.9 with TensorFlow, while CTGAN and
Copula GAN were implemented in Python 3.9 with PyTorch. More
powerful hardware, such as a state-of-the-art graphics processing
unit (GPU), would enable processing of the entire dataset together,
lifting the memory limitation of Section 4.2 and reduce training
duration.

5. Results and analysis

5.1. Data generation and testing

Vanilla GAN: As discussed in 4.4, the model was able to signifi-
cantly learn the data distributions but after 1880 epochs, $G$ no longer
improved in the learning process. A sample of 25 of the synthetic
data features was selected and compared with the same features in
the real data sample to assess the generation performance. Figure 2
plots the correlation between the selected data features for real data
in Figure 2(a), fake data in Figure 2(b) and the difference between
these in Figure 2(c). Relatively poor results may be observed since
the real and fake data differ significantly because mode collapse has
occurred (the typical Vanilla GAN flaw) with greatly dissimilar losses
at 3000 epochs of 0.0004 for $D$ and 7.86 for $G$. As introduced in 3.1,
Vanilla GAN aims to make both losses to converge at 0.5 to achieve
optimal generation. The training was continued beyond the number
of epochs when mode collapse appeared to occur ($\sim 1800$ epochs)
ensuring that this was indeed the case rather than merely learning
difficulty.

CTGAN: This GAN was initially trained without converting the
categorical features in the dataset. A second training run was con-
ducted using the same dataset with all categorical features converted
to numeric data types before fitting for the RDT Transformation.
The results differed slightly, as shown in Figure 3 for training with-
out conversion and Figure 4 for training with conversion. $D$ and
$G$ losses in the first case were 0.41 and $-2.4$ respectively, while
the corresponding losses for the second training run were 0.40
and 0.39. The first training instance was not able to attain optimal
generation as seen in the gap between $D$ and $G$. losses due to
nonconvergence and limited training epochs – increased training
epochs could improve the training convergence and performance.
The second training run, as indicated in the losses, implied train-
ing convergence and improved performance. This approach shows
that CTGAN can achieve higher performance with converted (all-
numeric) input data rather than unconverted (mixed categorical and
continuous) input data, which has been the norm with previous GAN
models.

Copula GAN: The data were loaded into the Copula GAN model
for training without converting the categorical features. Copula
GAN, being an improved version of CTGAN, showed better gen-
eration results as seen in Figure 5 with a sustained loss of 0.42 for
$D$ and 0.40 for $G$. This reveals that Copula GAN achieves similar
performance for mixed continuous and categorical input data, with
the second instance of CTGAN training using all-numeric input
data. This satisfied the study objective of generating realistic network
traffic data using a mixture of categorical and continuous features
as input data. Increased epochs could further decrease the losses


without losing convergence, thereby increasing the quality of the synthetic data.

5.2. Visualization of data characteristics

To summarize, visualize and compare the real and synthetic data clusters for each simulation experiment we first employed Principal Component Analysis (PCA) [30], which exploits data correlation. PCA reduces the dimensions of the data by finding new variables, the principal components, comprising linear combinations of the original variables obtained by projections onto an orthonormal base of the signal subspace. Figure 6 shows the principal components obtained by the different GANs for both real and fake data. Figure 6(a) shows that clusters between the real and fake data from Vanilla GAN PCA
Figure 5. Copula GAN results showing (a) the real data distribution, (b) the fake data distribution and (c) the difference between (b) and (a). When matched, the difference between the real and fake sampled distribution reduced significantly similar to the second instance of CTGAN training with only numeric input data. Thus, satisfying the study objective to attain optimal generation performance.

Figure 6. The first two PCA components for (a) Vanilla GAN; (b) first CTGAN training; (c) second CTGAN training; (d) Copula GAN.

do not have a similar shape. This can be attributed to the poor generation as reflected in the $D$ loss and the generation distribution disparity seen in Figure 2. PCA from the first and second CTGAN training (Figures 6(b) and (c)) and the Copula GAN training (Figure 6(d)) showed improved generation as the cluster shapes of the fake data have similar shapes to those of the real data.

We also considered the absolute values of the logarithmic means and standard deviations of the fake numerical data plotted against the real numerical data as shown in Figure 7. For the Vanilla GAN, there is poor correlation between these quantities (Figure 7(a)) but this improves markedly for CTGAN in Figures 7(b) and (c), and for Copula GAN in Figure 7(d). However, results from the second CTGAN training in Figure 7(c) that appear superior at first glance do not represent better performance on network traffic data generation since categorical features were converted to numerical values before training, meaning that the resulting synthetic data contain no non-numeric features.

The $D$ loss from the three models demonstrates that GANs can reproduce real network traffic data. However, there are variations in the generation levels between the CTGAN and the Copula GAN as the PCA in Figure 6(d) shows that clusters of the fake data from Copula GAN training bear a closer resemblance to the real data.
when visually compared with clusters of fake data from both CTGAN training runs.

5.3. Evaluation

We employed the Synthetic Data Evaluation Framework (Synthetic Data Vault – SDV 0.12.1) [29] to compare and evaluate the generated (synthetic) data with the real network traffic data for similarity, distance between real and generated distributions, and difficulty level in distinguishing the two data streams.

The metrics shown in Table 1 were obtained from the SDV and indicate that when categorical features are converted, CTGAN’s synthetic data have a higher similarity score (0.639) than the other architectures experimented with. This, as mentioned earlier, does not reflect the true performance of the models in generating genuine network traffic data since the data distribution is only numeric. Thus, it can be concluded that Copula GAN performs better, having the next higher similarity score (0.600). This performance surpasses results recorded by previous state-of-the-art architectures, including PcapGAN.

Considering the distribution of categorical features using a Chi-Squared test (CSTest), which indicates the probability that the sampled features are from the same distribution, CTGAN had a higher probability (0.995) than Copula GAN (0.923). A similar test but for continuous features using the Kolmogorov–Smirnov test (KSTest) showed that Copula GAN performed better (0.850) than CTGAN in both training runs; 0.631 (first training) and 0.847 (second training).

We also assessed the difficulty in distinguishing the synthetic data features from the real data features using the Support Vector Classification (SVC) detection method [31]. This uses a support vector machine to determine the best position of a hyperplane to distinguish data categories, here the synthetic features and the real features. The results here indicated the difficulty of distinguishing synthetic and real data. Here, only 28% of features from the Vanilla GAN could replace those in the real data without detection but this rose to 55% for CTGAN (69% when categorical features were converted) and some 85% for Copula GAN.

6. Discussion and conclusion

With various improvements to the initial framework, GANs have become a popular choice for many applications in various domains. While implementation in image, facial, anime and language, medical and multimedia domains are at advanced stages recording groundbreaking results, application in network traffic data generation is still in its infancy, albeit showing impressive and promising results with high potential.
Although the use of GANs for traffic generation has been limited in comparison to other application domains [2], there have been some significant successes such as PAC-GAN and Flow-Based Network Traffic Generation GAN. However, these methods are only able to train with converted categorical or non-numeric network traffic data features, limiting evaluation of accuracy and generated data similarity to the real data. The learning process for network traffic data distributions structured in tabular format is more demanding than when input data are used in an image format. This is largely due to multiple data types in the training data, data imbalance and multimodal or non-gaussian data distributions.

This research implements the CTGAN and Copula GAN variants to generate synthetic traffic data and evaluate them for overall similarity and feature per feature similarity. CTGAN uses the RDT hyper transformer and Copula GAN uses the Gaussian Copula Transformer, enabling them to simultaneously train with and generate categorical and continuous features. Results from both training runs are compared with results from Vanilla GAN using Yeo-Johnson’s power transformation to obtain Gaussian distributions for all features. The Vanilla GAN is unable to generate synthetic data that closely resemble the real data with significant disparity in the generated data. CTGAN and Copula GAN generate results that closely match the real data including the various data types matching those in the real distribution.

Further evaluation of the results show that Copula GAN performed better than CTGAN as more synthetic data features could pass as real data features without detection, a higher probability value for the KSTest, and higher similarity score when categorical features are not converted to numeric values prior to training. CTGAN, however, showed a higher probability value for CSTest showing it performed better in generating categorical features.

This study shows that GANs can reproduce realistic synthetic Network Traffic data despite the complexities associated with network traffic data distributions. Moreover, they can attain similar levels of success witnessed in other applications. In particular, Copula GAN is able to deliver synthetic data that closely resembles real data. This is shown by the close match between the false versus real logarithmic means and standard deviations, the form of the principal components resulting from PCA and the fact that some 85% of the features from the Copula GAN could replace those in the real data without detection. Further research in this domain is highly recommended as GANs have shown great potentials especially when further implemented to support network traffic synthesis for adversarial purposes, and Cyber and Intrusion Detection Systems. This study was also limited by system computational power, thus higher processing capacity would guarantee training with more data, more epochs and possibly improved performance.

Disclosure statement
No potential conflict of interest was reported by the author(s).

Funding
This research was supported by the School of Engineering, University of Warwick, United Kingdom.

Notes on contributors
Tertsegha J. Anande is currently a PhD student at the University of Warwick, Coventry, UK, researching the application of deep learning methods to network traffic to combat advanced persistent threats. He also works as a Senior Graduate Teaching Assistant in the Computer Science Department at the University of Warwick and is seconded from his role as a Lecturer in Nigeria, having previously worked as an Executive Director at Nobel Heights Global Services Limited. He has several published works in the areas of cyber security, computer simulation and data mining.

Sami Al-Saadi is a PhD student at the University of Warwick, Coventry, UK, investigating the application of deep learning to combat cyberattacks on software defined networks. Previously, he worked at Taibah University, Saudi Arabia, and received his MSc with Distinction in Computer Science from Coventry University in 2019.

Mark S. Leeson is a part-time Reader in the School of Engineering at the University of Warwick, UK, which he joined in 2000. From 1990 to 1992 he worked for NatWest Bank in London then held academic posts at the University of East London from 1992 to 1994, at Manchester Metropolitan University from 1994 to 1998 and at Brunel University from 1998 to 2000. His major research interests are optical communication systems and machine learning, with over 300 published works. He is a Senior Member of the IEEE and a Fellow the UK Institute of Physics.

ORCID
Mark S. Leeson http://orcid.org/0000-0003-0367-6228

References
and Privacy Workshops 2018; May 24, San Francisco, CA, p. 70–75. doi:10.1109/SPW.2018.00019.


