Distributed Generative Adversarial Networks for Anomaly Detection

Marc Katzef[1 [0000−0002−4229−3767]], Andrew C. Cullen[2 [0000−0001−8243−6470]], Tansu Alpcan[1 [0000−0002−7434−3239]], Christopher Leckie[2], and Justin Kopacz[3]

[1] Department of Electrical and Electronic Engineering, University of Melbourne, Australia
[2] School of Computing and Information Systems, University of Melbourne, Victoria, Australia
[3] Northrop Grumman Corporation, USA

Abstract. Cognitive radio networks can be used to detect anomalous and adversarial communications to achieve situational awareness on the radio frequency spectrum. This paper proposes a distributed anomaly detection scheme based on adversarially-trained data models. While many anomaly detection methods typically depend on a central decision-making server, our distributed approach makes better use of decentralized resources, and decreases reliance on a single point of failure. Using a novel combination of generative adversarial network (GAN) elements, participating cognitive radio devices learn a representation of local network activity data through a non-cooperative (strategic) game. Deviations from this expected network activity are flagged as anomalies and treated as possible network security threats, improving situational awareness. Tested on a range of time series datasets, the performance of the proposed distributed scheme matches that of state-of-the-art, centralized anomaly detection methods.

Keywords: Anomaly detection · Distributed · Generative adversarial networks · Cognitive radio networks

1 Introduction

Cognitive Radio (CR) is a communication architecture that strives to shift decision-making to network-connected devices themselves [1]. Originally envisioned to improve spectral efficiency (by intelligently hopping to unused frequencies), the devices composing CR Networks (CRNs) are autonomous, general-purpose computing resources. While the spectral efficiency aspect of CR has gathered great attention, the alternative benefits of device autonomy (such as increased situational awareness) are often overlooked.

Cognitive radio networks can be used to detect anomalous and adversarial communications to achieve situational awareness on the radio frequency spectrum. One of the objectives here is to identify potentially malicious communications by identifying unusual patterns. A secondary objective, as outlined in [2,3],

is to defend CRNs against a range of attacks spanning multiple network layers. These attacks include denial of service, firmware tampering, primary user emulation, false sensing reports, and spectrum data poisoning, as well as the standard wireless attacks of jamming and eavesdropping. In each of these attack scenarios, individual devices can collect unique, local data that can be used to identify unusual activity. The task of identifying these attacks has been treated as an anomaly detection problem and approached from different fields, as Section 2.1 explains.

Anomaly detection is the task of identifying outlying elements in a collection. Given a set of data samples (such as historical readings of network activity), anomaly detection can be formulated as the task of identifying the samples $x \in \mathbb{R}^d$ that were unlikely to be drawn from the same distribution as the remaining samples, $p_X(x)$. In a network security context, the samples being considered are typically feature vectors containing recent network traffic statistics, such as mean packet size, mean inter-arrival time and counts of each packet type. While many methods exist to approximate $p_X(x)$, few methods consider the resource constraints imposed by low-power devices in the CRN setting. With limitations on battery life, computation power, memory and storage, a CR device sharing all of its local observations with the entire CRN is infeasible. Shifting to a distributed anomaly detector would reduce the workload of any individual device and allow for arbitrary scalability.

This paper explores anomaly detection using GANs—adversarially-trained Machine Learning (ML) models with a strong data-generating ability and a high level of modularity—against test-time attacks. Recognizing that GANs learn the distribution of a given dataset, anomaly detection is a natural extension by identifying if a sample lies outside of the learned distribution. Recognizing that GANs are composed of several independent neural networks, exchanging these modules is a comparatively low-cost method to transfer learned experiences. Using these features, the contributions of this paper are (1) a game-theoretic evaluation and interpretation of a novel combination of GAN components (referred to as Peer-GAN), (2) a distributed formulation of Peer-GAN, applied to anomaly detection and (3) a framework for testing arbitrary anomaly detection methods in a distributed setting.

2 Related Work

Existing anomaly detection methods draw on a variety of related fields, with ML becoming the most prevalent in recent years. This section summarizes popular methods and one ML method in particular—the GAN.

2.1 Anomaly Detection

The task of anomaly detection has been explored using the fields of information theory, signal processing and a variety of ML methods. These methods all make a trade-off between storage requirements, computational requirements

and manual feature engineering. With minimal feature engineering and methods that balance storage and computation, ML has been a popular approach. When labelled training data is available, supervised ML methods often model anomaly detection as a binary classification problem. Supervised model-based works like [4] have used support vector machines to classify spectrum sensing fingerprints during a transmission. The approach in [5] used clustering and reinforcement learning. However, a significant performance improvement in terms of the Area Under Receiver Operating Characteristic (AUROC) curve was found by moving to the field of deep learning [6].

When signal class information is not known (e.g., when a network is under a novel attack), unsupervised ML methods may be used. As shown in [7], the field of unsupervised ML anomaly detection includes the well-established methods of k-Nearest Neighbor (kNN), isolation forest, feature bagging, Principal Component Analysis (PCA), auto-encoders and GANs. These methods can be divided into categories of distance-based (including kNN, isolation forests and feature bagging) and reconstruction-based (including PCA, auto-encoders and GANs). Distance-based methods measure some distance from a given sample to historical samples, which may be compared with a threshold to identify anomalies. For kNN, this distance is the norm of the difference between a sample and the k-th nearest historical sample. While the benchmarks in [7] show these methods perform well (in terms of AUROC), they depend on all N historical samples—giving $\mathcal{O}(N)$ storage requirements and $\mathcal{O}(\log(N))$ computational complexity for tree-based sample lookup. This storage requirement makes distance-based anomaly detection methods infeasible for use on storage-constrained devices such as CRs. To address these constraints, works like [8] have reduced the memory footprint of kNN by creating representative samples for each class. However, this decreased memory comes with decreased performance resulting in single hidden layer neural networks and support vector machines achieving higher classification accuracy.

The remaining, reconstruction-based anomaly detection methods do not depend on the entire training dataset for each subsequent classification. Instead, PCA and auto-encoders learn to reconstruct a given sample x to obtain $\hat{x}$ to then calculate a reconstruction error, $||x - \hat{x}||$, which may be compared with a threshold. These methods expect unfamiliar samples to give larger reconstruction errors. PCA (typically used as a dimensionality-reduction operation) reconstructs a sample by projecting a sample from the sample space to a lower-dimensional space (using a subset of the training dataset's eigen-vectors as the basis) and back. As these projections are two linear transformations, PCA reconstruction offers low fitting/training complexity and low storage requirements (only storing eigen-vectors) after training. Furthermore, [9] show that the PCA method may be approximated in a distributed environment. However, due to PCA's use of linear transformations, [9] states that PCA cannot represent data distributions with multiple data clusters.

Using deep neural networks (DNNs), auto-encoders overcome PCA's linearity constraint by performing arbitrary transformations to and from a low-dimensional auxiliary (code) space. After training, auto-encoders can update

their representation online (by training on additional samples) without the need to store the dataset after training. However, to initially form this representation, an auto-encoder must be trained on a centralized dataset, with collaboration hindered by the fact the auto-encoder's coded sample representation is arbitrary. As the following subsection explains, the GAN follows a similar pattern but the GAN imposes structure to the coded sample representation allowing for collaborative training.

2.2 GANs for Anomaly Detection

A GAN is composed of two ML models, the generator and discriminator, which are the players of a min-max game (see Section 3.1 for details). The tasks for the generator and discriminator are to, respectively, approximate $p_X(x)$ with a parameterized distribution, $p_G(x)$ and to discriminate between samples from $p_X(x)$ and $p_G(x)$. Existing works have applied GANs to the domain of anomaly detection by separating the GAN into its constituent components to achieve better performance than PCA, kNN and feature bagging on multiple datasets [10]. GANs have been applied to anomaly detection by either using discriminator output directly, measuring some distance between a sample x and $p_G(x)$, such as $\min_{\hat{x} \sim p_G} ||x - \hat{x}||$, or a combination of the two [10, 11]. One issue identified with using $p_G(x)$ is the large computation requirement to calculate the generator's anomaly score for individual samples. To speed up the required projection from sample space to the generator's output space, the Bidirectional GAN (BiGAN) is utilised in [12,13] to produce the *Efficient AnoGAN* and *Fast AnoGAN*. These BiGAN-based models introduce an additional player, the encoder, in the GAN game to learn the inverse to the generator's mapping, avoiding the need for an iteration-based projection to find $\hat{x}$.

Recognizing that GAN training involves separate, modular components—the players—recent research has considered collaborative configurations combining multiple generators (as in the MO-GAAL, [14]), multiple discriminators (as in the MD-GAN, [15]), or both. Previous studies [15–18] have shown that distributed GAN formulations learn a distribution that is closer to the real distribution (as measured by Fréchet inception distance) than that of a standalone GAN. When applied to anomaly detection in [16], a distributed GAN-based anomaly detector (using multiple discriminators and only discriminator-based anomaly scores) achieved a significantly higher accuracy at a given false positive rate.

3 Game-Theoretic Model of Generative Adversarial Networks

The GAN is a non-cooperative game between two ML models—a generator and a discriminator—in which the generator learns to approximate the distribution of a given dataset and the discriminator learns to distinguish between real data samples and the generator's synthetic output samples [19]. Through a zero-sum

(min-max) game between adversarially-trained ML models, a GAN can map points from an auxiliary data space to points from the same distribution as those in the given dataset as described in the following subsections.

3.1 GAN Games

The two independent ML models—a generator and a discriminator—are the players of the GAN engaged in a zero-sum non-cooperative (strategic) game denoted formally by the following tuple of players, action spaces and utility functions: $\mathcal{M} = \langle \{G, D\}, \{\Theta_G, \Theta_D\}, \{u_G, u_D\} \rangle$ where

- G is the generator player, which uses a θ_G-parametrized function approximator, $G(z; \theta_G)$ or $G(z)$ (for brevity) with $G : \mathbb{R}^l \mapsto \mathbb{R}^d$, to choose its actions,
- D is the discriminator player, which uses a θ_D-parametrized function approximator, $D(x; \theta_D)$ or $D(x)$ with $D : \mathbb{R}^d \mapsto \mathbb{R}$, to choose its actions,
- Θ_G is the generator's action/decision space, $\theta_G \in \Theta_G$.
- Θ_D is the discriminator's action/decision space, $\theta_D \in \Theta_D$,
- u_G and u_D are the generator and discriminator utility functions, respectively.

The typical choices for the above function approximators are DNNs, where Θ_G and Θ_D are the possible values for each network's weights. The selection of u_G and u_D define the type of GAN. In all GAN formulations, the goals of the generator and discriminator are to maximize their own utility function (or minimize their loss, the negative of utility). For the original GAN zero-sum game formulation and many subsequent variants, the player objective functions are $u_D = -u_G = u$ and take the form of

$$u(\theta_G, \theta_D) = \mathop{\mathbb{E}}_{x \sim p_X} \left[\phi(\hat{D}(x)) \right] + \mathop{\mathbb{E}}_{z \sim p_Z} \left[\phi(1 - \hat{D}(G(z))) \right].$$

Here $p_X(x)$ is the distribution of the given data samples, $x \in \mathbb{R}^d$, $p_Z(z)$ is the distribution of the noise samples for the generator (the generator's *latent space*), $z \in \mathbb{R}^l$, and ϕ is a measuring function.

In the original GAN definition (referred to as the min-max GAN), $\phi(x) = \log(x)$ and $\hat{D}(x) = \sigma(D(x))$ using the sigmoid function $\sigma(x) = (1 + e^{-x})^{-1}$ [19]. With these parameters, [19] shows that the shared utility function is equivalent to the Jensen-Shannon (JS) divergence $JS(p_G || p_X)$. In the Wasserstein GAN—a popular alternative to the min-max GAN—$\phi(x) = x$ and $\hat{D}(x) = D(x)$, which produces a shared objective function that is equivalent to the Wasserstein metric/earth-mover distance [20] (the significance of which is explained in Section 3.2).

To avoid the (computationally-intensive) projection method used in [11, 21] the BiGAN trains an additional ML model [22]. Instead of training one neural network (the generator) to learn $p_X(x)$, the BiGAN structure trains two neural networks (a generator and a θ_E-parameterized encoder, $E(x)$) simultaneously to collaboratively learn the conditional probability distributions

$p_G(x|z)$ and $p_E(z|x)$ that define the joint probability distributions $p_{GZ}(x,z) = p_G(x|z)p_Z(z)$ and $p_{EX}(x,z) = p_E(z|x)p_X(x)$. These distributions are learned in a version of the min-max GAN game where a modified discriminator, $D(x,z)$, discriminates over data-noise tuples, $(G(z),z)$ and $(x,E(x))$, from both adversaries. The standard-form game for the BiGAN is represented by the tuple $\langle\{G,E,D\},\{\Theta_G,\Theta_E,\Theta_D\},\{u_G,u_E,u_D\}\rangle$ where the objective function shared by all three neural networks is $u_D = -u_G = -u_E = u$, defined as

$$u(\theta_G,\theta_E,\theta_D) = \mathop{\mathbb{E}}_{x\sim p_X}\left[\phi(D(x,E(x)))\right] + \mathop{\mathbb{E}}_{z\sim p_Z}\left[\phi(1-D(G(z),z))\right].$$

In a BiGAN, the players have the complementary roles of generating synthetic data or noise, $(G(z),z)$ or $(x,E(x))$, and discriminating between (data, noise vector) tuples. These players pass data as shown in Fig. 1, where the generator receives input from an auxiliary space (the generator's latent space), the encoder receives data samples, $x \in p_X$ and the discriminator receives input from either the generator or the encoder.

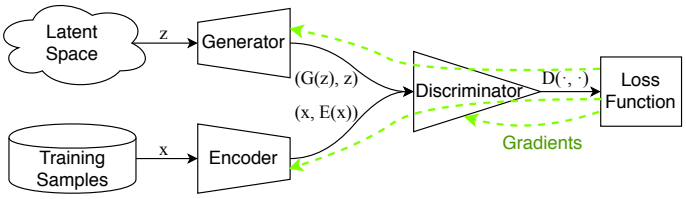


Fig. 1: ML model connections in the BiGAN game (with arrows labelled "Gradients" showing the path for back-propagation when updating player actions).

3.2 GAN Training Methods and Challenges

Using the BiGAN formulation, GAN training is the search for a Nash equilibrium, $(\theta_G^*,\theta_E^*,\theta_D^*)$, at the intersection of player best responses. One such Nash equilibrium is achieved when the generator's and encoder's joint probability distributions are equal, $p_{GZ}(x,z) = p_{EX}(x,z)$, and the discriminator can no longer distinguish between generator and encoder tuples. From [19], the discriminator model's output is

$$\sigma(D(x,z))\Big|_{x\sim p_X, z\sim p_Z} = \frac{p_{EX}(x,z)}{p_{GZ}(x,z)+p_{EX}(x,z)} = \frac{1}{2}. \tag{1}$$

Due to the non-convexity of GAN utility functions and the high-dimensional action spaces, finding a Nash equilibrium in GANs is challenging. The search for Nash equilibrium in a GAN game is performed by updating each player's action iteratively through stochastic gradient-based optimization to maximize their respective utilities. It has been shown that the use of two different learning rates in

player updates guarantees convergence to a Nash equilibrium [23]. However, [24] shows simply that (regardless of utility function) if one equilibrium exists, that equilibrium belongs to a family of equilibria achieved by permuting neurons to produce equivalent neural networks. With the existence of multiple Nash equilibria, a common problem in training GANs is convergence to sub-optimal solutions where the generator learns a small selection of samples (a problem termed *mode collapse*) or fails to learn any valid output before $\nabla_{\theta_G} u_G$ decays to 0 (a problem termed *vanishing gradient*).

The problem of vanishing gradients was identified in [19], prompting a revision to the generator's utility function (referred to as the non-saturating loss), in which the $\log(1-\sigma(D(\cdot)))$ term is replaced by $-\log(\sigma(D(\cdot)))$. In doing so, $\nabla_{\theta_G} u_G$ remains non-zero after improvements in the discriminator's action while leaving the equilibrium that is characterized by (1) unchanged. Another, more significant GAN modification is made in [20], which presented the Wasserstein loss for GANs to address mode collapse. This loss is shown to provide training gradients that vary smoothly with the difference between p_X and p_G, preventing the generator from getting trapped representing only a subset of data modes. While the Wasserstein loss-trained generators represent more modes of the training data, generators trained with the original min-max loss (and its non-saturating variant) generate higher fidelity samples of the modes they learn [25].

While the generator of a trained GAN can map noise samples, $z \sim p_Z$, to data samples, $x \sim p_G$, how these elements are linked is set arbitrarily during training. This arbitrary assignment is one additional hurdle for reconstruction using GANs. The study in [22] shows that in the presence of a perfect discriminator, the optimal generator and encoder are inverses (i.e., $E(G(z)) = z$ and $G(E(x)) = x$). Again due to the non-convexity of GAN utility functions, the BiGAN generator and encoder are unlikely to produce exact inverses in practice [22]. This problem is addressed by the GAN variants referred to as *CycleGAN* and *ALICE BiGAN* which add regularization terms based upon reconstruction error to u_G and u_E [26, 27]. CycleGAN implements this regularization with an $l1$-norm-based penalty in both directions, while ALICE BiGAN applies regularization only to the forward direction, but by using an additional, resource-intensive adversarially-trained model. These regularization terms ensure that player actions are updated such that $||G(E(x)) - x||$ and $||E(G(z)) - z||$ remain small.

4 A Novel Distributed GAN Framework

In this section, we present a novel GAN game combining non-saturating GAN loss for high-fidelity samples, Wasserstein loss to avoid mode collapse and ALICE BiGAN structure for cyclically-consistent mapping, referred to as *Peer-GAN*. As this section explains, this combination of existing GAN features are extended to a distributed training environment and used for anomaly detection.

4.1 Peer-GAN Game

Peer-GAN combines non-saturating loss, Wasserstein loss and ALICE BiGAN-like structure for a data flow given in Fig. 2. This combination of GAN ele-

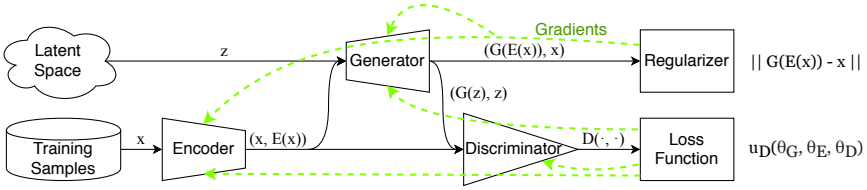


Fig. 2: Connections between ML models/players in the Peer-GAN game. Using BiGAN as a base, Peer-GAN adds a reconstruction-based regularizer in the generator and encoder loss functions to ensure G and E are inverses at game solutions.

ments targets accurate data representation, data reconstruction and straightforward extension to a distributed environment. Like BiGAN, the strategic-form game for a single instance, n, of Peer-GAN is represented by the tuple $\mathcal{M}_n = \langle \{G_n, E_n, D_n\}, \{\Theta_G, \Theta_E, \Theta_D\}, \{u_G, u_E, u_D\} \rangle$. Note that unlike classical GANs, this is a three player non-cooperative game with non-convex utility functions. In a distributed Peer-GAN, each instance comprises an independent generator, encoder and discriminator. Discriminator n, for example, chooses a strategy $\theta_{D,n} \in \Theta_{D,n}$ independently from $D_{i \neq n}$. The utility function for D_n,

$$u_D(\theta_G, \theta_E, \theta_D) = \mathop{\mathbb{E}}_{x \sim p_X} [\log(\sigma(D(x, E(x)))) + D(x, E(x))] + \mathop{\mathbb{E}}_{z \sim p_Z} [\log(1 - \sigma(D(G(z), z))) - D(G(z), z)], \quad (2)$$

is the sum of Wasserstein and non-saturating loss functions. With the remaining players both assigned the utility function, $u_G = u_E = u$, given as

$$u(\theta_G, \theta_E, \theta_D) = - \mathop{\mathbb{E}}_{x \sim p_X} [\log(\sigma(D(x, E(x)))) + D(x, E(x))] + \mathop{\mathbb{E}}_{z \sim p_Z} [\log(\sigma(D(G(z), z))) + D(G(z), z)] - \mathop{\mathbb{E}}_{x \sim p_X} [||G(E(x)) - x||_2] \quad (3)$$

which opposes u_D by having maxima where u_D has minima, while incorporating non-saturating generator loss and the expected data reconstruction error.

With utility functions as above, players G, E and D seek to maximize u_G, u_E and u_D. Combining utility functions and regularizers in this way alters some fixed points of the games mentioned in the previous sections. The one fixed point that remains unchanged is that of the desired Nash equilibrium—where $p_{GZ}(x, z) = p_{EX}(x, z)$ and $\mathbb{E}_{x \sim p_X, z \sim p_Z} [\sigma(D(x, z))] = \frac{1}{2}$. At this optimum, both the JS divergence and Wasserstein distance between p_{GZ} and p_{EX} are minimal and the discriminator can no longer discern between generator and encoder

samples. However, the regularization in these utilities adds an additional objective; at the global optimum, the generator and encoder are required to be exact inverses of each other on $\text{supp}(p_X)$. This property is the foundation of Peer-GAN-based anomaly detection, allowing for an anomaly score, $a_G(x)$, to be calculated as

$$a_G(x) = ||G(E(x)) - x||_2.$$

After evaluating $a_G(x)$ for a collection of test samples, a threshold may be chosen to classify future samples as anomalous. This threshold is typically chosen as n standard deviations greater than the mean $a_G(x)$ from the test set. Instead of defining a single threshold, our results in Section 5.3 capture anomaly detection performance for all possible thresholds using the AUROC.

4.2 Peer-GAN Distributed Training and Convergence

Using the Peer-GAN game definition above, the players G_n, E_n and D_n of a single Peer-GAN instance may be trained as described in Section 3.2. This section extends the formulation and analysis of Peer-GAN training to a distributed setting.

Distributed Training Peer-GAN's modular architecture allows for collaborative training in the form of parameter exchange. The proposed distribution strategy is to swap the discriminator player in each Peer-GAN instance with a neighboring instance after a chosen number of training steps while the generator and encoder remain in-place. As per Algorithm 1, in the proposed distributed framework, each of N participating devices is trained locally to determine $(\theta^*_{G,n}, \theta^*_{E,n}, \theta^*_{D,n})$ for local data, $p_{X,n}(x)$. For each of these obtained equilibria, the discriminator is no longer able to discern between generator and encoder output and adopts the strategy of outputting a constant

$$\mathbb{E}_{x\sim p_{X,n}}[\sigma(D_n(x, E(x)))] = \frac{1}{2}$$

as explained in Section 3.2. If the training data at device n is different from that at device $n+1$, $p_{X,n} \neq p_{X,n+1}$, the discriminator at device $n+1$ is expected to label unrecognized samples in $p_{X,n}$ as fake, giving

$$\mathbb{E}_{x\sim p_{X,n}}[\sigma(D_{n+1}(x, E(x)))] \leq \frac{1}{2}.$$

This discriminator output indicates that D_{n+1} has not yet/recently been trained to label $\text{supp}(p_{X,n})$ as real. The pairing $(\theta_{G,n}, \theta_{E,n}, \theta_{D,n+1})$ therefore gives G_n and E_n an opportunity to increase their respective utilities by outputting $p_G(x|z)$ and $p_E(z|x)$, the distributions learned by D_{n+1}. As performed in [16, 17] for the min-max GAN, the distributed Peer-GAN uses this pairing by swapping all discriminators in the framework such that $\mathcal{M}_n$ receives D_{n+1} after every T training epochs. This cyclic swapping scheme ensures that each discriminator

is trained on all available devices, avoiding any model bias from training on a subset of devices more than the others.

Algorithm 1: Distributed Peer-GAN training

Result: $(\theta_{G,n}, \theta_{E,n}, \theta_{D,n}),\ n = 1, 2, ..., N$
$\theta_{G,n}, \theta_{E,n}, \theta_{D,n} \leftarrow$ GlorotUniform, $n = 1, 2, ..., N$;
for *t = 1, 2, ..., num_epochs* **do**
  for *n = 1, 2, ..., N* **do**
    for *minibatch in minibatches* **do**
      $\theta_{G,n} \leftarrow \theta_{G,n} - \alpha_G \nabla_{\theta_G} u_G(\theta_{G,n}, \theta_{E,n}, \theta_{D,n})$;
      $\theta_{E,n} \leftarrow \theta_{E,n} - \alpha_E \nabla_{\theta_E} u_E(\theta_{G,n}, \theta_{E,n}, \theta_{D,n})$;
      $\theta_{D,n} \leftarrow \theta_{D,n} + \alpha_D \nabla_{\theta_D} u_D(\theta_{G,n}, \theta_{E,n}, \theta_{D,n})$;
    end
  end
  if $(t \mod T) = 0$ **then**
    tmp $\leftarrow \theta_{D,1}$;
    for *i = 1, 2, ..., N-1* **do**
      $\theta_{D,i} \leftarrow \theta_{D,i+1}$;
    end
    $\theta_{D,N} \leftarrow$ tmp;
  end
end

After a set number of training updates (or once training gradients drop below a chosen threshold) an optional final step is to measure anomaly detection performance on a held-out test set and broadcast the best-performing player parameters. In an environment where multiple Peer-GANs are being trained simultaneously, exchanging the current state of parameters between instances (as shown in Fig. 3) may be seen as both a method of compressed data transfer and as an additional form of regularization. This compressed data transfer view is from transmitting the result of training instead of the training data itself. The regularization view is from sudden changes in paired adversaries, which requires generalization for all models to maintain their current utility. The effect of this compression is shown in Table 1. By exchanging parameters instead of data samples, the required communication between devices (both transmission and reception) is altered and no longer dependent on the dataset's size. Where each device collects a large number of samples—which is expected from CRs—this reformulation keeps communication resource usage low and modifiable by changing the underlying discriminator model's size P and/or the exchange period T.

Training Convergence Distributed Peer-GAN training builds on centralized GAN training (in Section 3.2) by exchanging discriminator model parameters. These parameter exchanges may be seen as a compressed data transfer and a form of regularization, preventing overfitting to local data. This section explains the effect of these exchanges during training. Assuming each of N distributed Peer-GANs had reached an equilibrium before a parameter exchange, generator

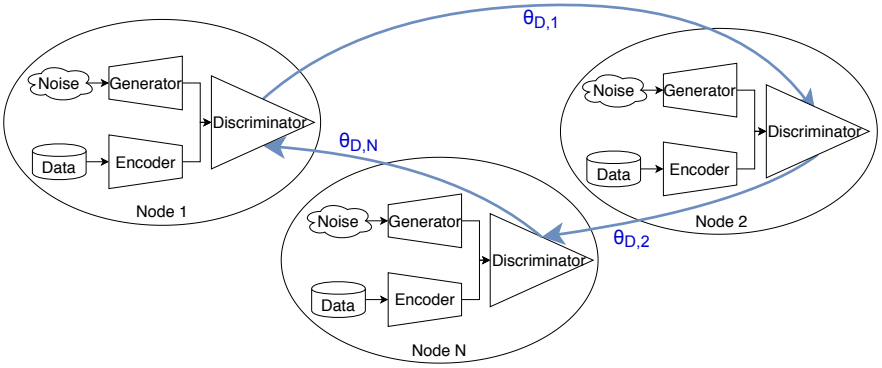


Fig. 3: Distributed Peer-GAN architecture, showing a cyclic swap of discriminator model parameters between three participating devices.

Table 1: The number of elements (e.g., floating point numbers) exchanged while training an anomaly detector using CRN-collected data where S is the dataset size, d is the sample length, N is the number of devices, P is the number of parameters in the model being exchanged, E is the number of training epochs, and T is the distributed strategy's swapping period.

Usage stat (per-device)	**Centralized**	**Distributed**
Transmitted data	$\frac{Sd}{N}$	$\left(\lfloor\frac{E}{T}\rfloor+1\right)P$
Received data	P	$\left(\lfloor\frac{E}{T}\rfloor+1\right)P$

and encoder pairs have learned conditional distributions of the joint distribution $p_{GZ,n}(x,z) = p_{EX,n}(x,z)$ and the generator and encoder conditional distributions match the distribution of that device's training data, $p_{G,n}(x|z) = p_{X,n}(x)$. After the first parameter exchange, the generator and encoder joint distribution no longer match the distribution of discriminator's training data, $p_{X,n+1}(x)$ and the new discriminator labels generator and encoder outputs using its previous strategy. Any samples from $\text{supp}(p_{G,n})\backslash\text{supp}(p_{X,n+1})$ are determined less likely to be real by D_{n+1}, worsening the generator's utility. However, any samples from $\text{supp}(p_{X,n+1})\backslash\text{supp}(p_{G,n})$ are opportunities for the generator to improve its utility.

For each player to obtain their optimal utility in a steady state, G_n must approximate the conditional probability

$$p_{G,n}(x|z) = \frac{1}{N}\sum_{i=1}^{N} p_{X,i}(x)$$

which is the mean of data distributions of all individual devices. At the same time, E_n must learn a representation of $p_{X,n}(x)$ that matches G_n's mapping

from the latent space. In this state, the optimal discriminator output is

$$\mathbb{E}_{x\sim p_X, z\sim p_Z}\left[\sigma(D_n(x,z))\right] = \frac{1}{2}$$

which, using (2), gives a discriminator loss (negative of utility) of

$$-u_D^* = -\left(\log\left(\frac{1}{2}\right) + 0 + \log\left(\frac{1}{2}\right) + 0\right) = \log(4).$$

Considering the first and second terms in (3) independently, the loss component contributed by G_n is

$$-u_G^* = -\mathop{\mathbb{E}}_{z\sim p_Z}\left[\log(\sigma(D_n(G_n(z),z))) + D_n(G_n(z),z)\right] = \log(2)$$

and the component contributed by E_n is

$$-u_E^* = \mathop{\mathbb{E}}_{x\sim p_{X,n}}\left[\log(\sigma(D_n(x,E_n(x)))) + D_n(x,E_n(x))\right] = -\log(2).$$

These are the ideal losses for each of the Peer-GAN players at a desired Nash equilibrium. In the limit as the discriminator swapping frequency increases, the distributed training scheme approaches the mini-batch training of N parallel Peer-GANs on $p_X(x)$, where all instances learn mappings between z and x that are compatible with other instances. Unfortunately, approaching this mini-batch training requires more frequent transmissions of $\theta_{D,n}$ and so a balance must be found between transmission overhead and distribution approximation accuracy.

5 Anomaly Detection and Simulation Results

To evaluate the performance of Peer-GAN as an anomaly detection system, a Peer-GAN was trained (along with its distributed counterpart) on a range of 1D datasets focusing on signals and compared with existing techniques. The proposed Peer-GAN anomaly detector and its distributed counterpart were implemented in Keras and Tensorflow using artificial neural networks for all three players, G, E and D, with ReLU as the activation functions for the two hidden layers in the generator and encoder, leaky ReLU for the three hidden layers in the discriminator, and linear activations for all model outputs. The Adam optimizer was used for all models, with learning rates set to $\alpha_G = \alpha_E = 10^{-4}$ and $\alpha_D = 10^{-5}$ for two time-scale separation allowing G and E to make use of D's learned modes after discriminator swaps. The datasets used here to evaluate Peer-GAN and other anomaly detectors are shown in Table 2.

To represent distributed network environments, the first two stated datasets contain measured data from simulated and real-world communication networks under attack. Each sample in the CRN dataset is a time-series window of received packet sizes, inter-arrival times and transmission durations averaged over five minute intervals in an OMNeT++ simulation (with occasional activity from

Table 2: Anomaly detection evaluation datasets.

Dataset	Sample count	Anomaly %	Feature count
CRN	4,190	50.0	111
KDD'99	445,372	11.0	121
MNIST	7,603	9.2	100
Synthetic	60,000	8.3	111

adversaries) [28]. In contrast, KDD'99 was collected from real-world network activity—from an Ethernet testbed exposed to attacks including denial of service and port scanning [29]. The samples in each of these datasets are vectors of features derived from received packet metadata (such as size, protocol and inter-arrival time) over discrete time windows. The MNIST dataset used here is a pre-processed version of the MNIST handwritten digit dataset where 100 significant features are retained from the original 28x28 images, available at [7]. Furthermore, this pre-processed dataset contains only the digit "0" as the typical sample, with samples from the "6" class added as anomalies. The final dataset (listed as *synthetic*) is a collection of randomly-generated time series signals with step changes and additive white Gaussian noise (with a higher power for generated anomalous samples). Each sample consists of 111 elements set at one of two distinct levels, flipping between the two at intervals of 35–40 indices, with additive noise with standard deviation of 10% and 50% of the difference between the two levels for typical and anomalous signals respectively. All sample elements were linearly shifted and scaled such that each feature in the training set fell in $[0, 1]$ and any categorical features were one-hot encoded before being embedded in feature vectors. Together, these datasets represent diverse use cases that could benefit from distributed anomaly detection.

Peer-GAN was evaluated by training two configurations; one with a single device which can access all data (the centralized case) and one consisting of three devices each with one third of the dataset (the distributed case). The distributed configuration trained each Peer-GAN with a fixed swapping interval of once per epoch. For effective anomaly detection, the desired trends during Peer-GAN training are:

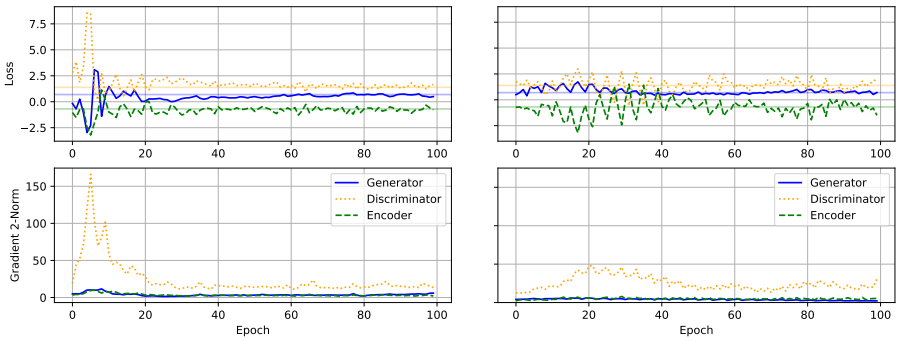
- All model losses converge to steady-state values stated in Section 4.2,
- All model loss gradients (as measured by $l2$-norm) decay to 0,
- Reconstruction error decreases for typical samples (seen as a low final value for G and E losses),
- Reconstruction error for typical samples becomes reliably smaller than that for anomalous samples (measured as AUROC using a dedicated test dataset).

The above training trends are addressed in the following subsections.

5.1 Peer-GAN Convergence

Throughout Peer-GAN training, each ML model's losses (negative of utility) and loss gradients were recorded for each parameter update step, for each device. Using the KDD'99 dataset for demonstration, the losses and gradients for the centralized and distributed cases (averaged over each epoch) are shown in Figures 4a and 4b respectively. In these tests, both Peer-GAN variants initially show large variations in losses (a typical feature of the Wasserstein training loss) but both converge to a relatively steady state after 30 training epochs (a state more stable than with Wasserstein loss by itself).

While the loss values shown in Figures 4a and 4b do not settle to constant values, the losses each remain in narrow bands. Notably, each of these bands match the expected losses explained in Section 4.2; $-u_D \approx \log(4)$, $u_G \approx \log(2)$ and $u_E \approx -\log(2)$. The visible instability in losses is expected to result from the use of stochastic gradient action updates for each player. Because of the inherent randomness in these update strategies, player loss gradients remain non-zero throughout training. However, Figures 4a and 4b show that these loss gradients decrease significantly after 20 training epochs, again indicating that a Peer-GAN game solution was found. One key difference between the two tested configurations is the height of each case's peak gradient. With a lower peak gradient, parameter swapping acts like regularization with training less prone to any player over-fitting and causing instability in other players' action updates. This better stability early in distributed training comes at the cost of a slight increase in variation later on—both effects of the initial model mismatch after a discriminator swap.



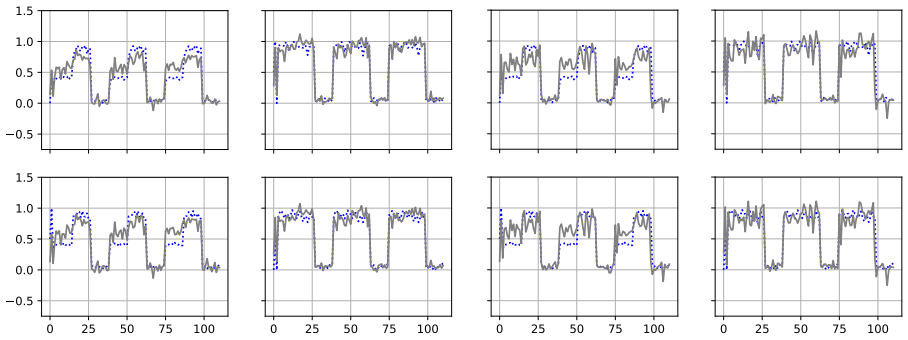
(a) Centralized Peer-GAN (b) Distributed Peer-GAN

Fig. 4: Player losses, $-u_P$, and gradients, $||\nabla_{\theta_P}||_2$, when training Peer-GAN on the KDD'99 dataset. After an initial period of instability, model losses approach the expected values from Section 4.2 (overlaid as solid, horizontal lines) indicating that a Nash equilibrium was found.

5.2 Sample Reconstruction

As Section 2.1 explains, the proposed Peer-GAN anomaly detector relies on sample reconstruction to identify anomalies. Reconstruction accuracy is therefore implicitly measured in anomaly detection performance. However, only the relative reconstruction accuracy is used to distinguish between samples from distributions. This section presents example reconstructions to observe Peer-GAN absolute reconstruction accuracy for both typical and anomalous samples.

Using the CRN dataset, random samples (both typical and anomalous) from the test set were passed through trained Peer-GAN models. Figures 5 and 6 show x values with $G(E(x))$ superimposed, where the ideal generator/encoder pair would satisfy $G(E(x)) = x, \forall x \in \text{supp}(p_X)$. From this small selection of typical samples, both the centralized and distributed Peer-GANs show accurate reconstruction in Fig. 5 (with neither model significantly outperforming the other). For anomalous samples (in Fig. 6) each model reverts to outputting noise (sometimes outside the bounds of the signal), mismatching the anomalous CRN signals. Together, these results indicate that Peer-GAN reconstructions are suitable for separating typical samples from anomalies and that distributed Peer-GANs perform similar reconstructions to centralized Peer-GANs.

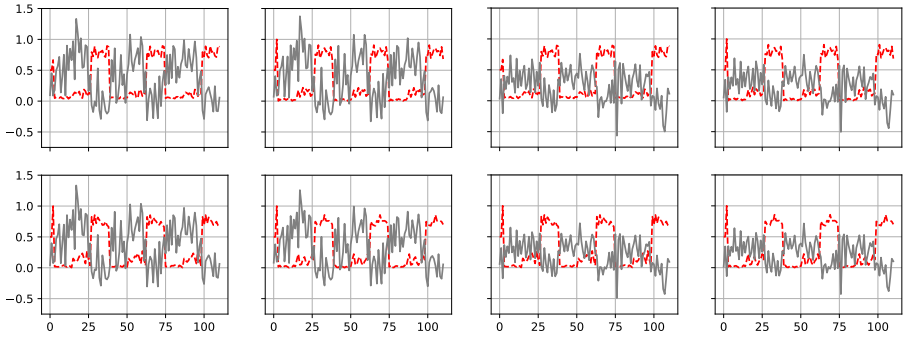


(a) Centralized Peer-GAN (b) Device 1 of a distributed Peer-GAN

Fig. 5: Peer-GAN reconstruction of typical CRN samples, $x \sim p_X$. Dotted lines represent the typical samples, solid lines the reconstructions, $G(E(x))$. For both models, the reconstruction visibly matches the given, typical samples.

5.3 Anomaly Detection Comparison

This section combines the previous section results of convergence and accurate reconstruction by applying Peer-GANs to anomaly detection. For each of the datasets in Table 2, the previously mentioned centralized Peer-GAN and distributed Peer-GAN were trained using shuffled and evenly-divided training



(a) Centralized Peer-GAN (b) Device 1 of a distributed Peer-GAN

Fig. 6: Peer-GAN reconstruction of anomalous CRN samples. Dashed lines represent the anomalous samples, solid lines the reconstructions, $G(E(x))$. Unlike the typical samples in Fig. 5, Peer-GAN reconstructions do not match anomalous samples.

datasets across participating devices. The shuffling in this dataset preparation breaks up any clusters of time-correlated data that may be present. Both Peer-GANs were trained using mini-batches of 500 samples for 100 epochs and (for the distributed Peer-GAN) a swapping period set as $T = 1$.

Performance Once trained, each Peer-GAN was evaluated by calculating AUROC on the complete test dataset. The highest AUROC of all devices was collected for three trials of both Peer-GANs. The means of these trials are presented in Table 3 along with the performance of existing methods. The competing methods shown are those with low feature engineering, storage and computational requirements as discussed in Section 2.1, using the default parameters from the python outlier detection library, [7] and the Efficient AnoGAN, [12].

As shown in Table 3, anomaly detection performance varied from reliably correct (an AUROC of 1.0) to reliably incorrect (an AUROC of 0.0). With an anomaly detector that randomly assigns labels expected to achieve an AUROC of 0.5, this performance range gives an insight into how each method handles different types of anomalies. For the CRN dataset, low performance is the result of the average energy in anomalous samples being lower than that of typical samples (as seen in Figures 5 and 6). For the CRN dataset, zero-mean, low-energy, random sample reconstructions would achieve an AUROC close to 0. Low performance on the synthetic dataset is the result of reconstructed sample noise exceeding the noise in typical test samples. For the remaining datasets, KDD'99 and MNIST, more sophisticated patterns must be learnt to reconstruct typical samples.

Of the tested methods, no one anomaly detector outperformed the competition for every dataset. For the CRN dataset, all anomaly detectors were

Table 3: Anomaly detection AUROCs for the distributed Peer-GAN and competing methods on the datasets listed in Table 2 with the highest AUROC per dataset in bold. The competing methods are Auto-Encoders (AE), Principal Component Analysis (PCA), MO-GAAL (MG), Efficient AnoGAN (EA), centralized Peer-GAN (C-Peer-GAN) and, separated as the only distributed anomaly detector, the distributed Peer-GAN (D-Peer-GAN).

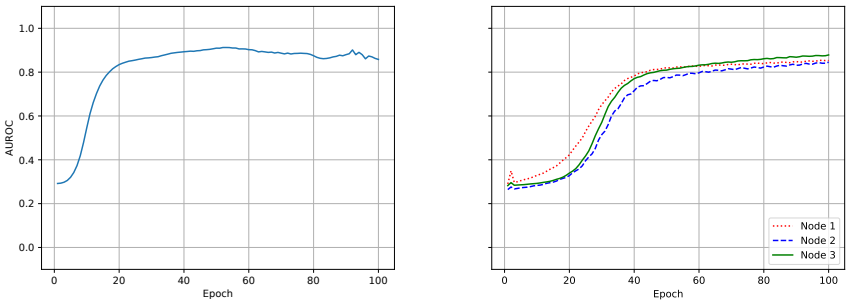
Dataset	AE	PCA	MG	EA	C-Peer-GAN	D-Peer-GAN
CRN	**1.0**	**1.0**	0.0	0.0	**1.0**	**1.0**
KDD'99	0.97	0.97	0.37	0.71	**0.99**	**0.99**
MNIST	**0.91**	**0.91**	0.55	0.29	0.83	0.88
Synthetic	0.51	0.51	0.19	0.54	0.98	**1.0**

either reliably correct or reliably incorrect for reasons explained above. For the synthetic dataset, the only anomaly detectors to perform better than random selection are the Peer-GAN variants. For the remaining datasets, auto-encoders, PCA and Peer-GAN variants achieved high performance. Compared to the remaining methods, both Peer-GAN variants were the only with AUROCs consistently above 0.51, showing a higher level of flexibility—being able to model both structural and fine-level detail in data samples—than PCA, auto-encoders and the remaining GAN-based methods. Notably, the performance of the distributed Peer-GAN was equal to or better than that of the centralized case. This counter-intuitive result may be explained by each Peer-GAN better learning the features of a small dataset—typically considered over-fitting—before sharing these results through discriminator exchanges.

Also tested were the methods kNN, feature bagging and isolation forest. With these methods, kNN was found to perform the best (or tied-best) across all datasets. kNN's high performance was followed closely by feature bagging, isolation forest and our distributed Peer-GAN. However, as explained in Section 2.1, the methods kNN, feature bagging and isolation forest have significant storage and computation requirements in use after training, making them unsuitable for resource-constrained devices.

Resource Usage Compared with the centralized methods of kNN, feature bagging and isolation forest, Peer-GAN does not need to load samples into memory for anomaly detection and replaces the transmission of data samples to a central server with transmission of model parameters. Using Table 1, the communication resource usage (summing transmissions and receptions) for each participating device in the distributed scheme is only 58% of a typical collaborative scheme for our discriminator model (where $P = 51,585$), the KDD'99 dataset (where $S = 445,372$ and $d = 121$), and three devices training for 100 epochs with a swapping period of 1 (giving $N = 3$, $E = 100$ and $T = 1$).

Training Stability As shown in Section 5.1, discriminator training gradients do not consistently decay to zero with additional Peer-GAN training epochs. The impact of these gradients was observed by evaluating each Peer-GAN's anomaly detection performance at the end of each training epoch. Results on the MNIST dataset (presented in Fig. 7) show that both centralized and distributed Peer-GAN performance remains stable in that their AUROC increases with additional training. The distributed Peer-GAN is seen to require additional training epochs to achieve the same performance as the centralized case. This delay is due to each distributed Peer-GAN training on a smaller dataset, which results in fewer parameter updates per epoch. Once identified, Peer-GAN retains its method of separating samples (in MNIST and additional datasets) after additional training epochs, making Peer-GAN suitable for on-line training—updating each Peer-GAN model as more data becomes available.



(a) Centralized Peer-GAN (b) Distributed Peer-GAN

Fig. 7: Peer-GAN anomaly detection AUROC during training on the MNIST dataset. For the distributed case, AUROCs are shown using solid, dotted and dashed lines for devices 1–3. In both cases, the Peer-GAN AUROC increases dramatically at early training stages, before levelling out.

6 Conclusion

In this paper, we presented Peer-GAN—a novel GAN-based anomaly detection scheme suitable for resource-constrained networked devices. Using a min-max game played locally on each device and collaboration between devices, Peer-GAN can accurately detect anomalies in 1D datasets while reducing data communication requirements for participating devices. Through theory and supporting simulation results, Peer-GAN was shown to converge to a Nash equilibrium wherein ML models—the players—achieve high reconstruction accuracy of dataset samples and, subsequently, anomaly detection performance to rival state-of-the art methods. In future work, we plan to investigate the effect of larger groups of devices and different swapping schemes (such as random pairing) in a distributed

Peer-GAN as well as Peer-GAN's robustness in the presence of device failure or adversaries.

Acknowledgements This work was supported in part by the Australian Research Council Linkage Project under the grant LP190101287, and by Northrop Grumman Mission Systems' University Research Program.

References

1. Joseph Mitola. An Integrated Agent Architecture for Software Defined Radio. *Ph.D. Dissertation, KTH*, July 2000.
2. Yi Shi, Tugba Erpek, Yalin E. Sagduyu, and Jason H. Li. Spectrum Data Poisoning with Adversarial Deep Learning. *arXiv:1901.09247 [cs]*, January 2019. arXiv: 1901.09247.
3. Ruiliang Chen, Jung-Min Park, and Jeffrey H. Reed. Defense against Primary User Emulation Attacks in Cognitive Radio Networks. *IEEE Journal on Selected Areas in Communications*, 26(1):25–37, January 2008.
4. Zhenxing Luo, Caiyi Lou, Shichuan Chen, Shilian Zheng, and Shaowei Li. Specific primary user sensing for wireless security in IEEE 802.22 network. In *2011 11th International Symposium on Communications Information Technologies (ISCIT)*, pages 18–22, October 2011.
5. Sundar Srinivasan, KB Shivakumar, and Muazzam Mohammad. Semi-supervised machine learning for primary user emulation attack detection and prevention through core-based analytics for cognitive radio networks. *International Journal of Distributed Sensor Networks*, 15(9):1550147719860365, September 2019.
6. Di Pu, Yuan Shi, Andrei V. Ilyashenko, and Alexander M. Wyglinski. Detecting Primary User Emulation Attack in Cognitive Radio Networks. In *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*, pages 1–5, December 2011. ISSN: 1930-529X.
7. Yue Zhao, Zain Nasrullah, and Zheng Li. PyOD: A Python Toolbox for Scalable Outlier Detection. 20:7, 2019.
8. Chirag Gupta, Arun Sai Suggala, Ankit Goyal, Harsha Vardhan Simhadri, Bhargavi Paranjape, Ashish Kumar, Saurabh Goyal, Raghavendra Udupa, Manik Varma, and Prateek Jain. ProtoNN: Compressed and Accurate kNN for Resource-scarce Devices. In *International Conference on Machine Learning*, pages 1331–1340. PMLR, July 2017. ISSN: 2640-3498.
9. Yuh-Jye Lee, Yi-Ren Yeh, and Yu-Chiang Frank Wang. Anomaly Detection via Online Oversampling Principal Component Analysis. *IEEE Transactions on Knowledge and Data Engineering*, 25(7):1460–1470, July 2013.
10. Dan Li, Dacheng Chen, Lei Shi, Baihong Jin, Jonathan Goh, and See-Kiong Ng. MAD-GAN: Multivariate Anomaly Detection for Time Series Data with Generative Adversarial Networks. *arXiv:1901.04997 [cs, stat]*, January 2019. arXiv: 1901.04997.
11. Thomas Schlegl, Philipp Seeböck, Sebastian M. Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery. *arXiv:1703.05921 [cs]*, March 2017. arXiv: 1703.05921.

12. Houssam Zenati, Chuan Sheng Foo, Bruno Lecouat, Gaurav Manek, and Vijay Ramaseshan Chandrasekhar. Efficient GAN-Based Anomaly Detection. *arXiv:1802.06222 [cs, stat]*, May 2019. arXiv: 1802.06222.
13. Thomas Schlegl, Philipp Seeböck, Sebastian M. Waldstein, Georg Langs, and Ursula Schmidt-Erfurth. f-AnoGAN: Fast unsupervised anomaly detection with generative adversarial networks. *Medical Image Analysis*, 54:30–44, May 2019.
14. Yezheng Liu, Zhe Li, Chong Zhou, Yuanchun Jiang, Jianshan Sun, Meng Wang, and Xiangnan He. Generative Adversarial Active Learning for Unsupervised Outlier Detection. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2019. Conference Name: IEEE Transactions on Knowledge and Data Engineering.
15. Corentin Hardy, Erwan Le Merrer, and Bruno Sericola. MD-GAN: Multi-Discriminator Generative Adversarial Networks for Distributed Datasets. *arXiv:1811.03850 [cs, stat]*, February 2019. arXiv: 1811.03850.
16. Aidin Ferdowsi and Walid Saad. Generative Adversarial Networks for Distributed Intrusion Detection in the Internet of Things. *arXiv:1906.00567 [cs, stat]*, June 2019. arXiv: 1906.00567.
17. Daniel Jiwoong Im, He Ma, Chris Dongjoo Kim, and Graham Taylor. Generative Adversarial Parallelization. *arXiv:1612.04021 [cs, stat]*, December 2016. arXiv: 1612.04021.
18. Quan Hoang, Tu Dinh Nguyen, Trung Le, and Dinh Phung. MGAN: Training Generative Adversarial Nets with Multiple Generators. page 24, 2018.
19. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *arXiv:1406.2661 [cs, stat]*, June 2014. arXiv: 1406.2661.
20. Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein Generative Adversarial Networks. In *International Conference on Machine Learning*, pages 214–223, July 2017.
21. Antonia Creswell and Anil A. Bharath. Inverting The Generator Of A Generative Adversarial Network (II). *arXiv:1802.05701 [cs]*, February 2018. arXiv: 1802.05701.
22. Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial Feature Learning. *arXiv:1605.09782 [cs, stat]*, April 2017. arXiv: 1605.09782.
23. Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. *arXiv:1706.08500 [cs, stat]*, June 2017. arXiv: 1706.08500.
24. D. Thierens. Non-redundant genetic coding of neural networks. In *IEEE International Conference on Evolutionary Computation*, pages 571–575, May 1996.
25. Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are GANs Created Equal? A Large-Scale Study. *arXiv:1711.10337 [cs, stat]*, October 2018. arXiv: 1711.10337.
26. Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *arXiv:1703.10593 [cs]*, November 2018. arXiv: 1703.10593.
27. Chunyuan Li, Hao Liu, Changyou Chen, Yunchen Pu, Liqun Chen, Ricardo Henao, and Lawrence Carin. ALICE: Towards Understanding Adversarial Learning for Joint Distribution Matching. *arXiv:1709.01215 [cs, stat]*, November 2017. arXiv: 1709.01215.
28. Sandamal Weerasinghe. sandamal/omnet_simulation, 2020. Available: https://github.com/sandamal/omnet_simulation. [Accessed: 10-June-2020].
29. The UCI KDD Archive. KDD Cup 1999 Data, October 1999.