Estimate the Implicit Likelihoods of GANs with Application to Anomaly Detection

Shaogang Ren, Dingcheng Li, Zhixin Zhou, Ping Li
Cognitive Computing Lab
Baidu Research
10900 NE 8th St. Bellevue, WA 98004, USA
{shaogangren, lidingcheng, zhixinzhou, liping11}@baidu.com

ABSTRACT

The thriving of deep models and generative models provides approaches to model high dimensional distributions. Generative adversarial networks (GANs) can approximate data distributions and generate data samples from the learned data manifolds as well. In this paper, we propose an approach to estimate the implicit likelihoods of GAN models. A stable inverse function of the generator can be learned with the help of a variance network of the generator. The local variance of the sample distribution can be approximated by the normalized distance in the latent space. Simulation studies and likelihood testing on real-world data sets validate the proposed algorithm, which outperforms several baseline methods in these tasks. The proposed method has been further applied to anomaly detection. Experiments show that the method can achieve state-of-the-art anomaly detection performance on real-world data sets.

CCS CONCEPTS
- Theory of computation → Unsupervised learning and clustering.

KEYWORDS
anomaly detection, generative adversarial networks, unsupervised learning, density estimation

ACM Reference Format:

1 INTRODUCTION

Many real-world high dimensional data sets concentrate around low dimensional unknown manifolds. Deep models provide new approaches to estimate the density of extreme high dimensional data. Generative models, e.g., generative adversarial networks (GANs) [13], can learn the distributions of high dimensional data sets and generate samples as well. GANs use the adversarial loss as their training objective, which penalizes dissimilarity between the distribution of the generated samples and the real samples. Given infinite approximation power, the original GAN objectives aim to minimize the Jensen-Shannon divergence between the real data distribution and generated samples.

The generator in a GAN can be seen as a nonlinear parametric mapping \( g : Z \rightarrow X \) from the low dimensional latent space to the data manifold. With the realistic pictures generated with GANs [6, 17], one can claim that a well regularized GAN models can approximate the true data manifolds well. With good data manifold approximations, people try to leverage the GANs to other tasks beyond image generation, e.g., anomaly detection [1, 30], photo style transformation [38], and text generation [34].

Due to the broad applications of GANs, it is important to measure the distribution likelihoods of any given samples. However, GANs are implicit models, and it means the sample likelihoods cannot be computed directly. The discriminator used in the GAN is designed to classify the samples from true data distribution and the samples from the generator. Thus the discriminator cannot estimate the likelihoods of the samples that do not belong to either distributions. It is proved by [5] that there exists an inverse function of the generator that can project samples in the sample space into the latent space. [29] provide an approach to estimate the spectral implicit density of generative models. However, this method does not provide a qualitative way to estimate the likelihood of any given samples. In this paper, we bring up a method that can learn an inverse function of the generator of GAN models that preserve the measurement consistency, and then apply the inverse function to data sample likelihood estimation. In the following subsections, we first review some manifold concepts for deep generative models, and then we briefly introduce neural network models for density estimation. In the last subsection, we present a short survey on the application of generative models to anomaly detection.

1.1 Deep Generative Models as Manifolds

Recently, people have tried to apply manifold analysis to generative models. For example, [22] improves generative training by adding manifold invariant into the classifier. The authors of [33] try to perform geodesic clustering with deep generative models. In [4], the authors add stochastic variables to the VAE generator. With the variance network, the metric estimation on the data manifold can be improved. Mathematically, a deterministic generative model \( x = g(z) \) can be seen as a surface model if the generator \( g \) is sufficiently smooth.

Here, we briefly review the basic concepts on surfaces. A deep generative model represents an embedding function, \( g : Z \rightarrow X \), from a low-dimensional latent space \( Z \subseteq R^d \) to a submanifold \( M \subseteq R^D \). Usually we have \( D \gg d \). We assume \( g \) is a smooth injective mapping, so that \( M \) is an embedded manifold. The Jacobian of \( g \) at
\[ z \in \mathcal{Z}, \quad J_g(z), \quad \text{providing a mapping from the tangent space at } z \in \mathcal{Z} \]
\[ \text{into the tangent space at } x = g(z) \in \mathcal{X}, \text{ i.e., } J_g : T_z\mathcal{Z} \rightarrow T_x\mathcal{X}. \]

This mapping is not surjective and the range of \( J_g \) is restricted to the tangent space of the manifold \( \mathcal{M} \) at \( x = g(z) \), denoted as \( T_x\mathcal{M} \).

Since GAN can generate realistic images, \( \mathcal{M} \) is close to the true data manifold, namely \( M_{\text{data}} \).

We represent the Riemannian metric as a symmetric, positive definite matrix field, \( \mathcal{M}(z) \), defined at each point \( z \) on the latent coordinate space, \( \mathcal{Z} \). \( \mathcal{M}(z) \) is given by the following formula:

\[ \mathcal{M}(z) = J_g(z)^\top J_g(z). \]

Given two tangent vectors \( u, v \in T_z\mathcal{Z} \) in coordinates, their inner product is defined as \( \langle u, v \rangle = u^\top \mathcal{M}(z)v \). Let us consider a smooth curve \( \gamma : [a, b] \rightarrow \mathcal{Z} \).

This corresponds to a curve on the manifold, \( g \circ \gamma(t) \in \mathcal{M} \). The arc length of \( \gamma \) is given by the following formula:

\[ L(\gamma) = \int_a^b \sqrt{\dot{\gamma}(t)^\top \mathcal{M}(\gamma(t))\dot{\gamma}(t)} \, dt, \]

where \( \dot{\gamma}(t) \) is the first order derivative of \( \gamma \) at time \( t \). For example, with the explicit formula of geodesic distance, we can then apply geodesic clustering on the data manifold [12, 33].

### 1.2 Change of Variable

When \( \dim(\mathcal{Z}) < \dim(\mathcal{X}) \), the change of variable theorem is known in the context of geometric measure theory as the smooth coarea formula [16, 20]. With the assumption that \( q \) is a topological embedding from \( \mathcal{Z} \) to the submanifold of \( \mathcal{X} \). It says that

\[ p_X(x) = p_Z(g^{-1}(x)) \det (J_g(g^{-1}(x))^\top J_g(g^{-1}(x)))^{-\frac{1}{2}}, \]

where we assume the prior distribution on the latent space \( p_Z \) is \( N(0, I_d) \) throughout this paper. For \( x \in \mathcal{M} \), we have \( z = g^{-1}(x) \), and we obtain

\[ \log \{ p_X(g(z)) \} = \log \{ p_Z(z) \} - \frac{1}{2} \log \left( \det (J_g(z)^\top J_g(z)) \right). \]  

(1)

For a real-world data set, typically we would not know the true tangent space dimension number of \( M_{\text{data}} \) at a given data sample \( x \). People usually set \( d = \dim \mathcal{Z} \) larger than the \( T_x M_{\text{data}} \) for all \( x \). This leads to the problem that \( J_g \) does not have a full rank, and so does \( \mathcal{M}(z) \). It means that it is impossible to directly apply Eq. (1) with logarithm of zero determinant to estimate the sample density.

### 1.3 Density Estimation with Neural Networks

Along with the thriving of deep neural networks and GANs, a variety of density estimation methods have been developed. The MADE model [11] is proposed to estimate the likelihood values with masked neural networks. RealNVP [8] and Glow [18] are generative models that can estimate samples’ likelihood values. Both RealNVP and Glow employ reversible neural networks with special Jacobian metric whose determinant can be easily calculated. Similar to RealNVP and Glow, FLJord [14] is a generative model which can yield sample likelihood values with free-form Jacobian reversible neural network. FlowGAN [15] estimates data distribution by combining maximum likelihood estimation (MLE) and adversarial loss. The combination of MLE and adversarial loss may reduce the quality of generated samples. Most of these models use flow models to avoid the singularity issues coming with GANs. However, using the same dimension number for both the latent space and the input space may violate the fact that most real-world data sets are following distributions on low dimensional manifolds. As discussed in previous subsections, GANs can approximate the real data distribution in a more reasonable approach. In this paper, we try to develop a method that can estimate the quantitative likelihood or density values of data samples by leveraging the manifold approximation power of GANs.

Note that, even without likelihood estimation, people already apply GANs to anomaly detection. We will also apply our GAN likelihood estimation method to anomaly detection and demonstrate its effectiveness by comparing with existing anomaly detection methods. A review on anomaly detection with generative models is given in the next subsection.

### 1.4 Anomaly Detection with Generative Models

Generative models have been taken as effective ways to learn data representations and have been applied to anomaly detection. The auto-encoder based approaches [2, 37] first train a model that can reconstruct normal samples and then use the reconstruction errors to identify abnormal samples. The authors of [39] assume the latent spaces of data sets follow Gaussian mixture distributions. Instead of utilizing auto-encoder based methods that derive statistical anomaly criterion in light of data distributions or energies, they employ GAN-based framework to detect abnormal medical images in [30].

In their work, the latent variables of the samples are inferred with stochastic gradient descent. The anomaly score is estimated with the reconstruction error from the inferred latent variables in addition to the discrimination value from the discriminator of GAN model. Different from these existing methods, we try to develop a new anomaly detection method based on the proposed GAN likelihood estimation. In the sense of generativeness, our approach shares something in common with [30].

In summary, we aim to propose a framework that can estimate the log-likelihood of samples by leveraging the GAN models. The proposed method can approximate the local variance of any given data points with the help of two learned networks, the variance for the generator and the inference network for the latent representation. With the variance network, the singularity of the generator’s Jacobian matrix can be avoided. Thus, the likelihoods of samples can be calculated with the Riemannian metric. Our experiments on several real-world data sets reveal that, in the tasks of anomaly detection and likelihood testing, our approach considerably outperforms other baseline methods.

### 2 METHODOLOGY

GANs attempt to estimate the empirical distributions of high dimensional data sets. Given the learned generator \( g \), and a latent variable \( z \), we can estimate the generated sample \( g(z) \)’s log-likelihood with Eq. (1). The computation of Eq. (1) implies the full rank of \( g \)’s Jacobian matrix regarding \( z \). We can use a low dimensional latent space, i.e., a small \( d \), to obtain a full rank Jacobian matrix for almost any \( z \in \mathcal{Z} \). However, experimentally a small \( d \) may hurt the performance of GANs. In this paper, we try to resolve these issues with an extension of the classical GAN model. Our primary goal is to estimate the likelihoods of any given data samples. We need to learn an inference network to directly map a given data point \( x \) in the input space to a latent variable \( z \) that can be mapped back to a point...
close to \( x \) on the manifold \( M \). Then the sample \( x \)'s log-likelihood value can be computed thanks to the generator and the proposed variance network.

### 2.1 The Variance Network of the Generator

We add a variance network \( \sigma \) to the generator \( g \), and it extends the generator to a stochastic one,
\[
f(z) = g(z) + \sigma(z) \odot \epsilon, \quad \text{and} \quad g : \mathcal{Z} \to \mathcal{X}, \sigma : \mathcal{Z} \to \mathbb{R}^d_+ , \epsilon \sim N(0, I_d).
\]

Here \( g \) is the mean function, \( \sigma \) is the variance function, and \( \odot \) stands for element-wise multiplication. For a given \( \epsilon \), \( f \) is an approximation of \( g \) and \( E_{e \sim N(0,1_d)} f(z) = g(z) \). The variance network \( \sigma \) extends the generator \( g \) to the whole input space \( \mathbb{R}^d_+ \). We will show that the singularity issue of the \( g \)'s Jacobian matrix can be overcome with its variance network \( \sigma \). To ensure that the variances are large in the regions without or less data samples, we formulate the distance from \( z \) to all the cluster centers with learned kernel weights. This step can be done in the prepossessing stage and it can avoid computation overhead to the main algorithm.

The RBF function \( \sigma \) function is given by,
\[
\sigma(z) = \left( W^2 v(z) \right)^{-\frac{1}{2}},
\]
where \( a \) is a hyper-parameter for the kernel, and \( W \) is the network parameter need be learned from the training data samples. The samples with larger distance from the cluster centers will have larger variances. Given a generator network \( g \), we can learn the variance network weights \( W \) by minimizing the distance between \( f(z) \) and \( x \). With the stochastic generator \( f \) and the variance function, the Riemannian metric [4] can be written as
\[
\bar{M}_f = E_{e \sim N(0,1_d)} J_f^T(z)J_f(z)
= J_f^T(z)J_f(z) + J_f^T(z)J_f(z).
\]

We have the following lemma for the estimation of likelihoods.

**Lemma 1.** With \( K \geq \dim(\mathcal{Z}) + 1 \) and a full rank \( W^2 \), \( \bar{M}_f \) is a full rank matrix. The log-likelihood of the a generated sample is given by
\[
E_{e \sim N(0,1_d)} \log(p_X(f(z))) = \log(p_{\mathcal{Z}}(z)) - \frac{1}{2} \log(\det(\bar{M}_f)).
\]

### 2.2 The Inference Network Learning

For a given data sample \( x \), we need to find the corresponding latent space representation \( z \) before we can use Eq. (3) and Eq. (4) to compute the Riemannian metric and thus the likelihood value. As shown in Figure 1, we try to learn an approximate inverse of the generator \( g \) and the variance function, \( \epsilon \). Given a \( \epsilon \) and a full rank \( J_f(z) \) at \( z \) and \( e \), \( f \) is locally invertible in the open neighborhood \( f(S), S \) being an open neighborhood of \( z \). We try to lean a \( h : f(S) \to S \) such that
\[
E_{e \sim N(0,1_d)} \log(p_X(f(z))) = \log(p_{\mathcal{Z}}(z)) - \frac{1}{2} \log(\det(\bar{M}_f)).
\]

Here \( \lambda_i, 0 \leq i \leq d \) are the eigenvalues of \( \bar{M}_f \).

**Remark:** Note that
\[
E_{e \sim N(0,1_d)} \log(p_X(f(z))) = \log(p_{\mathcal{Z}}(z)) - \frac{1}{2} \log(\det(\bar{M}_f)) = \log(p_{\mathcal{Z}}(z)) - \frac{1}{2} \sum_{i=1}^d \log(\lambda_i).
\]
learn the parameters of $\sigma$ and $h$,
\[
\max_{h,\sigma} \mathbb{E}_{x \sim \mathcal{P}_{\text{data}}(x)} \left[ \log(p(x|z)) | z=h(x) \right].
\] (5)

Here $(p(x|z))$ is parametric with the extended generator $f$. According to Eq. (2), given a $z$, $p(x|z)$ is a multivariate Gaussian distribution with mean $\mu(z)$, and co-variance matrix $\Lambda_{\sigma}$, and $\Lambda_{\sigma}(\cdot) = \text{diag}(\sigma(z))$. We add a constraint to Eq. (5) to include the prior distribution of $z$ into the objective, and it forces the posterior $q(z|x)$ ($z = h(x)$, $x \sim \mathcal{P}_{\text{data}}(x)$) to be close to the prior of $z$, $p(z)$ ($p(z)$ is the $p_Z$ in section 1.2).

\[
\max_{h,\sigma} \mathbb{E}_{x \sim \mathcal{P}_{\text{data}}(x), z=h(x)} \left[ \log(p(x|z)) - \text{KL}(q(z|x)||p(z)) \right].
\]

Each element of the empirical distribution $q(z|x)$ is a Gaussian distribution with the sample average of $h(x)$ as the mean, and the standard deviation of $h(x)$, $x \sim \mathcal{P}_{\text{data}}(x)$ as the variance. The objective formula is different from the VAE [19] model. In our model we do not model the variance of distribution $q(z|x)$ in order to reduce model complexity. Moreover, given the generator $g$, our objective is to learn a variance network $\sigma$ for the generator $g$ rather than the decoder in VAEs. It is easy to prove that, with $z \in \mathbb{Z}$, and $\epsilon \sim \mathcal{N}(0, I_D)$, $f$ follows a Gaussian Process with $g$ as the mean function, and $\Lambda_{\sigma}$ as the covariance function, and $\Lambda_{\sigma}$ is a diagonal matrix with $\sigma$ along the diagonal.

**Lemma 2.** Assuming $g$ is a well learned generator function with GAN model, with $z \in \mathbb{Z}$ and $\epsilon \sim \mathcal{N}(0, I_D)$, the stochastic generator $f(z) = g(z) + \sigma(z) \odot \epsilon$ can be taken as $f(z) \sim \mathcal{GP}(g(z), \Lambda_{\sigma})$.

**Proof.** Any collection of the entries of $f$ follows a Gaussian distribution. According to the definition of Gaussian process, $f(z)$ follows Gaussian Process.

By stacking $f$ and $h$ together, it is a regression model to reconstruct a data sample $x$ ($x \sim \mathcal{P}_{\text{data}}$), i.e., $\hat{x} = g(h(x)) + \sigma(h(x)) \odot \epsilon$. With the inferred latent variables and a collection of reconstructed data samples $\hat{x} \sim f(h(x))$, the whole framework is similar to the Gaussian Process latent variable model (GP-LVM, [23]). The objective for $h$ and $\sigma$ becomes
\[
L_{h,\sigma} = \mathbb{E}_{x \sim \mathcal{P}_{\text{data}}} \left[ \sum_{i=1}^{D} \left( \frac{1}{2} \left( g_i(h(x)) - x_i \right)^2 / \sigma_i^2(h(x)) \right) - \log \sigma_i(h(x)) - \text{KL}(q(z|x)||p(z)) \right].
\]

### 2.3 Stabilize Training

Experimentally, it is rather difficult to learn a pair of perfect $h$ and $\sigma$ for a given $g$. We integrate the learning of both networks with the training of GANs. We have the following objective to adjust the generator $g$ and $h$,
\[
\min_{g,h} \mathbb{E}_{z \sim p(z)} \left[ \| z - h(g(z)) \|^2 \right].
\]

It is similar to the regularization term in InfoGAN [7] that enforces mutual information between the generated samples and partial latent variables. VEEGAN [32] also included this term to their regularization to reduce model collapses. Experiments show that with this term the model can converge fast and the training process is more stable. Figure 1 shows the scheme of the $g$ and $h$.

### 2.4 Algorithm

The algorithm to learn $h$ and $\sigma$ is combined with the learning of the generator $g$ and the discriminator $d$ in GAN. The complete procedure is given in Algorithm 1.

**Algorithm 1 The proposed algorithm to learn GAN with inference net $h$ and variance net $\sigma$**

1. **Input:** Data sample set
2. **Output:** Networks $g$, $d$, $h$, and $\sigma$
3. Sample a fixed number of hidden variables from $P_Z$ for $K$-means clustering; Compute $\lambda_k, c_k, k = 1, ..., K$;
4. while Training not finished do
5. Sample minibatch $m$ samples $(x^{(1)}, ..., x^{(m)})$ from $P_{\text{data}}$;
6. Sample minibatch $m$ samples $(z^{(1)}, ..., z^{(m)})$ from noise prior $P_Z$;
7. Updating the discriminator $d$ with ascending the gradient
   \[
   \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log d(x^{(i)}) + \log(1 - d(g(z^{(i)}))) \right].
   \]
8. Sample minibatch $m$ samples $(z^{(1)}, ..., z^{(m)})$ from noise prior $P_Z$;
9. Updating the generator $g$ by descending the gradient
   \[
   \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \left[ \log(1 - d(g(z^{(i)}))) \right].
   \]
10. Sample minibatch $m$ samples $(x^{(1)}, ..., x^{(m)})$ from $P_{\text{data}}$;
11. Updating $h$ and $\sigma$ by ascending the gradient $\nabla_{\theta_{h,\sigma}} L_{h,\sigma}$;
12. Sample minibatch $m$ samples $(z^{(1)}, ..., z^{(m)})$ from noise prior $P_Z$;
13. Updating $h$ and $g$ by descending the gradient
   \[
   \nabla_{\theta_{h,g}} \frac{1}{m} \sum_{i=1}^{m} \| z^{(i)} - h(g(z^{(i)})) \|^2.
   \]
14. end while

Let $W_t$ denote $W$ at iteration $t$ in Algorithm 1. We have the following Lemma 3 regarding the updating steps for $W$. Lemma 3 shows that with a randomly initialized $W$, we almost can ensure each $W_t, t = 1, ..., T$ is with full rank, and thus a full rank Jacobian matrix for the stochastic generator $f$. This means we can
we will get $u$ from $J$ almost with full rank due to the randomness of training samples.

LEMMA 3. Let $W^t$ be the matrix $W$ at the updating step $t$ of algorithm 1, then $W^{t+1}$ will be the element-wise multiplication of $W^t$ with another matrix.

PROOF. Given a sample $x$, the latent variable is $z = h(x)$. We have the loss regarding $x$,

$$L^x_{h, \sigma} = \frac{1}{2} \sum_{i=1}^{D} \left[ -\frac{1}{2} g_i(h(x)) - x_i^2 / \sigma_i^2(h(x)) - \log \sigma_i(h(x)) \right] - \text{KL}(q(z|x)||p(z)).$$

For the $i$th entry of $x$, the gradient regarding $W$ is

$$\frac{\partial L^x_{h, \sigma}}{\partial W} = -\frac{1}{2} \frac{g_i(h(x)) - x_i^2}{\sigma_i^2(h(x))} \hat{\beta}_i \otimes v^T(z).$$

Here $\hat{\beta}_i(z) = W_i \cdot v(z)$. Let

$$u_i(x) = \frac{\hat{\beta}_i^{-1}}{\sigma_i(h(x))} \left( 1 - \frac{g_i(h(x)) - x_i^2}{\sigma_i^2(h(x))} \right).$$

we get

$$\frac{\partial L^x_{h, \sigma}}{\partial W} = W \otimes (u(x)v^T(z)).$$

Here $u(x) = [u_1(x), ..., u_D(x)]^T$.

Without losing the generalization, we ignore the learning rate. For $t$, we have

$$W^t = W^{t-1} + W^{t-1} \otimes (u^{t-1}(x)(v^{t-1}(z))^T)$$

$$= W^{t-1} \otimes \left( 1 + u^{t-1}(x)(v^{t-1}(z))^T \right).$$

With a batch of training samples $B$, $W^t$ will be the element-wise product of $W^{t-1}$ with an aggregated updating matrix, which is

$$W^t = W^{t-1} \otimes \left( 1 + \frac{1}{|B|} \sum_{x \in B, z = h(x)} u^{t-1}(x)(v^{t-1}(z))^T \right).$$

This concludes the lemma. $\square$

Let $W^0$ be the initial value of $W$, the final step $W^T$ will be the element-wise multiplication of $W^0$ with the aggregated updating matrix regarding all steps. We are almost sure about the full rank of $W^T$ as $W^0$ is randomly initialized and the updating matrix is almost full rank due to the randomness of training samples. The time consuming part is the computation of the likelihood in the testing stage. It is due to the calculation of $J_g$, and $g$ is a mapping from $\text{dim}(Z)$ to $\text{dim}(X)$.

3 THEORETICAL ANALYSIS

Algorithm 1 basically follows the same procedure to learn the generator $g$ and the discriminator $d$ in GANs. The only interaction between $g$ and $h$ is to enhance the mutual information between latent variables and the generated samples. The loss functions for $d$ and $g$ can be adapted to any other forms of divergence to measure distribution distances, such as Wasserstein distance. Without loss generality, we study the theoretical upper bound of distribution estimation with the proposed approach. We can follow existing approaches to analyze the estimation upper bound. We adopt the definition of Earth Moving Distance, or equivalently Wasserstein-1 distance from [3].

DEFINITION 1. For distributions $P$ and $Q$ on $\mathbb{R}^d$, Wasserstein-1 distance between them $W(P, Q) = \inf_{\gamma \in \Pi(Q, P)} \mathbb{E}_{(X, Y) \sim \gamma} \|X - Y\|_1$.

In the following theorems, $P^{(n)}$ denotes the empirical distributions of $P$ with $n$ samples. In other words, suppose $x_1, x_2, ..., x_n$ are i.i.d. samples of $P$, then $P^{(n)} = \frac{1}{n} \sum_{i=1}^{n} 1(x = x_i)$ is a empirical distribution of $P^{(n)}$. For distributions $P$ and $Q$, $P + Q$ denotes the distribution of $X + Y$ for $X \sim P$ and $Y \sim Q$ independently. Suppose $Q^{(n)} = \frac{1}{n} \sum_{i=1}^{n} 1(y = y_i)$ is a empirical distribution of $Q^{(n)}$, then $P^{(n)} + Q^{(n)} = \frac{1}{n} \sum_{i=1}^{n} 1(z = x_i + y_i)$ is an empirical distribution of $P + Q$. It is worth noting that $P^{(n)} + Q^{(n)} \neq \frac{1}{n} \sum_{i=1}^{n} 1(z = x_i + y_i)$.

THEOREM 1. Let $F$ be certain function class and $g : \mathbb{R}^d \rightarrow \mathbb{R}^d$ be an $L$-Lipschitz function for $D > d > 2$. Let $Z$ be random variable satisfying $\mathbb{E}[|Z|^2] \leq \epsilon$ and $P_1$ be the distribution of $g(Z)$. Let $P^{(n)}$ be an $n$-sample empirical distribution of $g(Z) + \epsilon$, where $Z$ follows distribution such that $\mathbb{E}[|Z|^2] \leq \alpha$ and $\epsilon \sim N(0, \text{diag}(\sigma_1, ..., \sigma_D) / \sqrt{D})$, and

$$\hat{g} = \arg\min_{\hat{g} \in \mathcal{F}, \hat{g}(Z) = \hat{g}} W(P, P^{(n)}).$$

Let $\hat{P}$ be the distribution of $\hat{g}(Z)$, then

$$\mathbb{E}[W(\hat{P}, P_1)] \leq 2C_d \sqrt{\epsilon} \mathbb{E}[|Z|^2] + 5 \sqrt{\sum_{i=1}^{D} \frac{\sigma_i^2}{D}}$$

for the constant $C_d$ only depending on $d$ from Theorem 2.

PROOF. Let $P, P_1$ and $\hat{P}$ be the distribution of $g(Z) + \epsilon, g(Z)$ and $\hat{g}(Z)$ respectively. By triangle inequality,

$$W(\hat{P}, P_1) \leq W(\hat{P}, P^{(n)}) + W(P^{(n)}, P) + W(P, P_1).$$

The first term on the RHS can be bounded by

$$W(\hat{P}, P^{(n)}) = \min_{\hat{g} \in \mathcal{F}, \hat{g}(Z) = \hat{g}} W(\hat{P}, P^{(n)}) \leq W(P, P^{(n)}),$$

where the inequality is due to the fact that $g \in \mathcal{F}$ and $g(Z) \sim P$. Hence, we have

$$W(\hat{P}, P_1) \leq 2W(P^{(n)}, P) + W(P, P_1).$$

It is sufficient to bound these two terms to obtain the desired result. We decompose $P^{(n)} = P_1^{(n)} + P_2^{(n)}$, where $P_1^{(n)}$ and $P_2^{(n)}$ are $n$-sample
We combine these bounds and obtain
\[ W(\mathbb{P}_1, \mathbb{P}_2) \leq W(\mathbb{P}_1, \mathbb{Q}) + W(\mathbb{P}_2, \mathbb{Q}) + W(\mathbb{P}_1, \mathbb{P}_2). \]

By Lemma 4 and Jensen’s inequality, we have
\[ W(\mathbb{P}_1, \mathbb{P}_2) = W(\mathbb{P}_1 + \mathbb{P}_2, \mathbb{P}_1) \leq \mathbb{E}_{\mathbb{P}_2}[\|\epsilon\|] \leq (\mathbb{E}_{\mathbb{P}_2}[\|\epsilon\|^2])^{1/2} \leq \sqrt{\frac{\sum_{i=1}^{D} \sigma_i^2}{D}}. \]

Since \( g \) is an \( L \)-Lipschitz function, Let \( Z \sim \mathcal{Q}(n) \) be an \( n \)-sample empirical distribution of \( Z \). Suppose \( \mathcal{Q}(n) = \frac{1}{n} \sum_{i=1}^{n} \delta_z \) and \( \mathbb{P}_n = \frac{1}{n} \sum_{i=1}^{n} 1\{x = g(z_i)\} \). Since \( g \) is an \( L \)-Lipschitz function, we have
\[ W(\mathbb{P}_n, \mathbb{Q}(n)) \leq LW(\mathbb{Q}(n)). \]

By Theorem 2, we have
\[ W(\mathbb{P}_1, \mathbb{P}_2) \leq LW(\mathbb{Q}(n)) \leq C_d \sqrt{\mathcal{N}n^{-d}}. \]

By 5,
\[ \mathbb{E}[W(\mathbb{P}_1, \mathbb{P}_2)] = \mathbb{E}[W(\mathbb{P}_1, \mathbb{P}_2) + W(\mathbb{P}_2, \mathbb{Q})] \leq \mathbb{E}_{\mathbb{P}_2}[\|\epsilon\|] \leq \sqrt{\frac{\sum_{i=1}^{D} \sigma_i^2}{D}}. \]

We combine these bounds and obtain
\[ \mathbb{E}[W(\mathbb{P}_1, \mathbb{P}_2)] \leq 3W(\mathbb{P}_1, \mathbb{P}_2) + 2\mathbb{E}[W(\mathbb{P}_1, \mathbb{P}_2) + W(\mathbb{P}_1, \mathbb{P}_2)] \leq 3\sqrt{\frac{\sum_{i=1}^{D} \sigma_i^2}{D}} + 2C_d \sqrt{\mathcal{N}n^{-d}} \leq 2C_d \sqrt{\mathcal{N}n^{-d}} + 5\sqrt{\frac{\sum_{i=1}^{D} \sigma_i^2}{D}}, \]
as desired.

**Lemma 5.** Let \( \mathbb{P} \) and \( \mathbb{Q} \) be distributions on \( \mathbb{R}^d \). Let \( \mathbb{P}(n) \) and \( \mathbb{Q}(n) \) be the empirical distribution of \( \mathbb{P} \) and \( \mathbb{Q} \) respectively. Then
\[ \mathbb{E}[W(\mathbb{P}(n), \mathbb{Q}(n))] \leq EX-Q[\|X\|]. \]

**Proof.** We will use the definition \( \mathbb{P}(n) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}(x = x_i) \) and \( \mathbb{Q}(n) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}(y = y_i) \) and \( \mathbb{P}(n) + \mathbb{Q}(n) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}(z = x_i + y_i) \).

Let \( y \in \mathbb{P}(n), \mathbb{Q}(n) \) such that \( (x, y) \sim y \), then \( \mathbb{P}(y = x_i + y_i) = 1 \). Then
\[ W(\mathbb{P}(n), \mathbb{Q}(n)) \leq \mathbb{E}_{(x, y) \sim y}[\|X - Y\|] \leq \frac{1}{n} \sum_{i=1}^{n} \|x_i - (x_i + y_i)\| = \frac{1}{n} \sum_{i=1}^{n} \|y_i\|. \]

We have \( \frac{1}{n} \sum_{i=1}^{n} \|y_i\| = EX-Q[\|X\|] \) as desired.

We will introduce a useful theorem from [10], which provides the bound between empirical distribution and original distribution in Wasserstein distance. We refer readers to find the proof in the original paper.

**Theorem 2 ([10], Theorem 1).** Let \( \mathbb{P} \) be distribution on \( \mathbb{R}^d \) and \( \mathbb{P}_n \) be its \( n \)-sample empirical distribution. Suppose \( \mathbb{E}_{\mathbb{P}}[\|X\|^2] \leq c \), then
\[ \mathbb{E}[W(\mathbb{P}, \mathbb{P}_n)] \leq C_d \sqrt{\mathcal{N}n^{-d}}, \]

for some constant \( C_d \) only depending on \( d \).

## 4 EXPERIMENTS

In this section, we first evaluate the proposed likelihood estimation method with simulated and real-world data sets. Then we try to apply the proposed method to anomaly detection tasks. In the following experiments, we use 'InvGAN' to represent the proposed model. The implementation of the proposed InvGAN is based on the PaddlePaddle platform.

### 4.1 Likelihood Estimation on Synthetic Data

In this subsection, we investigate the proposed approach with simulated data. We have two iid latent variables \( z_1, z_2 \) following \( N(0, 1) \) distribution. The samples are simulated with \( X = [\sin z_1, \cos z_2, \sin z_1 \sin z_2, \cos z_2, \sin z_2 \cos (z_1 + z_2)] + \epsilon \), and each entry of \( \epsilon \) follow \( N(0, 0.01) \). We can easily compute the Jacobian regarding \( X \) and \( z \) to get the ground truth likelihood value for each simulated data sample with Eq. (4). The simulated data set contains 50,000 samples for training and 50,000 samples for testing. In the proposed

[1]https://www.paddlepaddle.org.cn/
model, the generator and the discriminator both have two fully connected layers with 30 hidden nodes, and the invert function $h$ has the same structure as the discriminator except the output dimension.

We compare the proposed model with two density estimation models FlowGAN, and FlowVAE. FlowGAN [15] is a hybrid model with maximum likelihood estimation (MLE) and adversarial loss. They employ the coupling layers proposed in [8] for the generator. With the coupling layers, the generator has the same number of dimensions for both latent space and input space. With the coupling layers [8], the determinate of the generator can be easily computed, and thus the likelihood of the samples. FlowGAN [15] train the generator based on both MLE and the adversarial loss functions.

We build a FlowVAE model with the decoder that has the same structure as the generator in the FlowGAN model. InvGAN and FlowGAN have the sample structure for the discriminator, which includes four linear layers and three Relu layers. We use the same batch size and epoch number to train three models.

As stated previously, the data set is simulated with 2 latent variables, and the dimension of the input (sample) space is 6. Since FlowGAN and FlowVAE use the same dimension number for both input space and the latent space, this may result in a constant offset between the model estimations and the ground truth likelihood. To fairly compare different models in figures, we add a constant value to the likelihood values of both FlowGAN and FlowVAE. Figure 2 gives the comparison between the ground truth and the estimated likelihood values for the tested set with three methods. We can see that compared with FlowGAN and FlowVAE, the proposed model can give more smooth estimations for the likelihood values, and improved fitting with the ground truth likelihood curve.

### 4.2 Likelihood Estimation on Real Data Sets

<table>
<thead>
<tr>
<th>Model</th>
<th>MNIST</th>
<th>CIFAR10</th>
<th>ImageNet32</th>
</tr>
</thead>
<tbody>
<tr>
<td>RealNVP</td>
<td>1.06</td>
<td>3.49</td>
<td>4.28</td>
</tr>
<tr>
<td>Glow</td>
<td>1.05</td>
<td>3.35</td>
<td>4.09</td>
</tr>
<tr>
<td>FFJORD</td>
<td>0.99</td>
<td>3.40</td>
<td>-</td>
</tr>
<tr>
<td>MADE</td>
<td>2.04</td>
<td>5.67</td>
<td>-</td>
</tr>
<tr>
<td>MAF</td>
<td>1.89</td>
<td>4.31</td>
<td>-</td>
</tr>
<tr>
<td>FlowGAN</td>
<td>3.26</td>
<td>4.21</td>
<td>-</td>
</tr>
<tr>
<td>InvGAN</td>
<td>1.29</td>
<td>1.23</td>
<td>1.02</td>
</tr>
</tbody>
</table>

We compare our model with the other likelihood estimation methods using three real-world data sets, MNIST, CIFAR10, and ImageNet32. The methods listed in the Table 1 include RealNVP [8], Glow [18], FFJord [14], FlowGAN [15], MADE [11], and MAF [26]. Most of these methods are based on maximum likelihood estimation (MLE). FlowGAN is a hybrid model combining MLE and adversarial loss. RealNVP, Glow, and FlowGAN rely on reversible coupling layers to preserve the density mass between the input space to latent space. Different from invertible neural network models, MADE [11] and MAF [26] are mask-based neural density estimation methods. For this set of experiments we use three convolution layers and one linear layer for the generator, the discriminator, and the inference network. More details about the network structures are given in Appendix A.

![Figure 3: The left plot shows the average negative log-likelihood values of a MNIST testing image set using InvGAN at different latent space dimension ($d$). The right plot presents the corresponding bits/dim for the testing set under the revised metric at different $d$ values.](image)

For “InvGAN”, different number of dim($Z$) may lead to different values of log-likelihood values even for the same data sample (the left plot in Figure 3). The bits per dimension metric [14, 15], i.e., $-\log p_X(x)/D, x \in R^D$, may not apply to our model, as the proposed model always yields smallest bits/dim value due the small latent space dimension number used in our model. To fairly compare with the other models, we propose to use the a revised version of bits per dimension metric in the comparison, which is $-\log p_Z(z)/d + \frac{1}{2} \log (|\det(J_{\theta}(z))^\top J_{\theta}(z)))/D$. Here $d$ is the latent space dimension size. With the new metric, the bits/dim of InvGAN

---

**Table 1: Bits/dim on testing data sets of MNIST, CIFAR10, and ImageNet32 for different models; lower is better**
can be bounded in a reasonable range (as shown in the right plot of Figure 3). The new metric can compensate the difference resulted from different latent space dimension numbers. For flow or reversible neural network based density estimation methods, the value of bits per dim does change as they take $D = d$. We follow the experimental setup in FFJord [14] and FlowGAN [15], and Table 1 compares different methods under the new metric on testing data sets. We use the likelihood values presented in FFJord [14] and FlowGAN [15]. We can see that even with the revised metric, our model can consistently perform better compared with other models.

4.3 Anomaly Detection

**Table 2: Statistics of the data sets for anomaly detection**

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Feature</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>1,024</td>
<td>60,000</td>
</tr>
<tr>
<td>CIFAR10</td>
<td>3,072</td>
<td>60,000</td>
</tr>
<tr>
<td>Arrhythmia</td>
<td>274</td>
<td>452</td>
</tr>
</tbody>
</table>

In this subsection, we apply the proposed likelihood estimation to three anomaly detection tasks. The data sets we use for anomaly detection are MNIST [24], CIFAR10 [21], and Arrhythmia [9]. Table 2 gives more details on the three data sets. For all of the results, our anomaly score is defined similar to [1]. With the log-likelihood value set, $S = \{s_j : \text{LLK}(x_j), x_j \in T\}$, the anomaly score is

$$s_i' = \frac{s_i - \min S}{\max(S) - \min S}.$$  

Here $T$ is the testing set. The anomaly methods we compared our method with are given in next subsection.

4.3.1 Baseline Methods.

**Deep Structured Energy Based Models (DSEBM)** [36] are energy-based models for anomaly detection. Similarly to denoising autoencoder, the main idea is to accumulate the energy across layers of neural networks. Two types of anomaly scoring criteria were proposed: energy based and reconstruction error based. In our experimental results, DSEBM-r represents results with reconstruction error and DSEBM-e are results with energy based approach.

**Deep Autoencoding Gaussian Mixture Model (DAGMM)** [39] is an autoencoder-based method for anomaly detection, and it can achieve state-of-the-art results. In DAGMM, an auto-encoder as well as an estimator network are trained jointly. The auto-encoder is to generate latent space representations, and the estimator is to output parameters of a GMM modeling the lower-dimensional latent space. The likelihood value of a sample’s latent representation computed with the learned GMM is taken as the anomaly detection metric.

**AnoGAN** [30] is a GAN-based method for anomaly detection. The method trains a GAN model to recover a latent representation for each test data sample. The anomaly score is calculated with a combination of the reconstruction error and the discrimination score from the discriminator network. The reconstruction error measures how well the GAN can reconstruct the data via the inferred latent variable and the generator. The authors of [30] compare the two components for the anomaly score and we picked the variant which performed best in the paper.

**Efficient GAN-Based Anomaly Detection (EGBAD)** [35] is another anomaly detection method based on GAN. Different from AnoGAN [30], their model learns an inverse function of the generator to speed up the inference of the latent variable. Similar to AnoGAN, The anomaly score in this approach includes two parts: the fitting error from reconstruction and the discrimination score.

**GANomaly** [1] uses a conditional generative adversarial network that jointly learns the generation of high-dimensional image space and the inference of latent space. Employing encoder-decoder-encoder sub-networks in the generator enables the model to map the input image to a lower dimension vector, which is then used to reconstruct the generated output image. The use of the additional encoder network maps this generated image to its latent representation. Minimizing the distance between these images and the latent vectors during training aids in learning the effective representations for the normal samples.

**Adversarially Learned Anomaly Detection (ALAD)** [35] is a recently proposed GAN based anomaly detection method. Similar to EGBAD [36], ALAD learns an encoding function to infer the latent variable for testing samples. Their model is enhanced with three discriminators. With a cycle consistency between sample space and latent space, the training of their model can be stabilized in learning representations for anomaly detection.

**One Class Support Vector Machines (OC-SVM)** [31] are a classic kernel method for anomaly detection and density estimation that learns a decision boundary around normal examples. The radial basis function (RBF) kernel is employed in the experiments. The $\nu$ parameter is set to the expected anomaly proportion in the data set, which is assumed to be known, whereas the $\gamma$ parameter is set to be inversely proportional to the number of input features.

**Isolation Forests (IF)** [25] are a classic machine learning technique that isolates abnormal samples rather than learning the distribution of normal data. The method constructs trees across randomly chosen features according to randomly split values. The anomaly score is defined as the average path length from a testing sample to the root. The experiments results for this model are obtained with the implementation in the scikit-learn [27] package.

4.3.2 MNIST. We take one class of the ten numbers in the MNIST [24] data set as the abnormal class, and the rest classes as the normal class. In the training phase, we use only the images from nine normal classes to train GAN and our model. In the testing state, we use the images from all ten classes in the testing set. The experiment setup is following GANomaly [1]. The results for GANomaly [1] is obtained by running the online code with the optimal parameters given in the script. The results for the other methods are based on the result section in [1]. The upper plot in Figure 4 shows the results of all of the methods on ten classes. The proposed method outperforms the others methods in all the ten tasks.

4.3.3 CIFAR10. Similar to the MNIST data set, we take one class as the abnormal class, and the rest classes as normal. We follow the experimental setup in [1]. The testing involves samples from both the normal classes and the abnormal class in the testing data
Figure 4: Anomaly detection with different models on the MNIST and CIFAR10 data sets.

As shown in the lower plot in Figure 4, our method leads the best results in all ten anomaly detection tasks, especially for the class dog. The anomaly detection results on two real data sets show that our model can perform well on the variance estimation for the distribution samples and outliers.

Table 3: Anomaly detection on MNIST and CIFAR10

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Model</th>
<th>AUROC</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>VAE</td>
<td>0.3420 ±0.1861</td>
</tr>
<tr>
<td></td>
<td>AnoGAN</td>
<td>0.4390 ±0.1116</td>
</tr>
<tr>
<td></td>
<td>GANomaly</td>
<td>0.7390 ±0.0882</td>
</tr>
<tr>
<td></td>
<td>InvGAN</td>
<td>0.7920 ±0.0786</td>
</tr>
<tr>
<td>CIFAR10</td>
<td>OC-SVM*</td>
<td>0.5843 ±0.0956</td>
</tr>
<tr>
<td></td>
<td>IF*</td>
<td>0.6025 ±0.1040</td>
</tr>
<tr>
<td></td>
<td>DSEBM-r*</td>
<td>0.6071 ±0.1007</td>
</tr>
<tr>
<td></td>
<td>DSEBM-e*</td>
<td>0.5956 ±0.1151</td>
</tr>
<tr>
<td></td>
<td>ALAD*</td>
<td>0.6072 ±0.1201</td>
</tr>
<tr>
<td></td>
<td>AnoGAN</td>
<td>0.5949 ±0.1076</td>
</tr>
<tr>
<td></td>
<td>EGBAD</td>
<td>0.4450 ±0.0786</td>
</tr>
<tr>
<td></td>
<td>GANomaly</td>
<td>0.6065 ±0.0390</td>
</tr>
<tr>
<td></td>
<td>InvGAN</td>
<td>0.6391 ±0.0608</td>
</tr>
</tbody>
</table>

Table 3 presents comparison among different anomaly detection methods for both image data sets. The experimental setups for AnoGAN, EGBAD, GANomaly, and the proposed InvGAN are the same and follow [1]. In order to obtain a comprehensive comparison with existing methods, we also include some anomaly detection results for CIFAR-10 from [35] (models marked with * in Table 3).

We notice that there are some small experimental differences between [35] and what we used for CIFAR-10. In [35], they take one class any the normal one, and the rest classes as the abnormal ones. Anomaly detection is essentially a binary classification problem. The numerical results in Table 3 can validate the significance of the proposed model. In summary, the proposed method outperforms the other auto-encoder and GAN based approaches for both image data sets. More details about the implementation of the networks are given in Appendix A.

4.3.4 Arrhythmia. We further investigate the method on a tabular data set. The Arrhythmia [9] data set is obtained from the ODDS repository. The smallest classes, including 3, 4, 5, 7, 8, 9, 14, and 15, are combined to form the anomaly class, and the rest of the classes are combined to form the normal class. Table 4 shows the anomaly detection results with different methods. Due to the small training set, classic methods outperform deep generative models on precision and F1 score. However, our method achieves the highest recall values and a high F1 score as well. With the help of the variance network, our method is relatively robust compared with other deep neural network methods. More details about the implementation of the InvGAN can be found in Appendix B. Figure 5 gives the convergence of the loss $L_{h, \sigma}$ for different data sets.

Figure 5: $L_{h, \sigma}$ at different iterations for different data sets.
5 CONCLUSIONS

In this paper, we propose an approach to estimate the implicit likelihoods of GANs. By leveraging an inference function and a variance network of the generator, the likelihoods of testing samples can be estimated. Simulation study and likelihood testing on real-world data sets validate the advantages of the proposed method. We further apply the method to three anomaly detection tasks. Experimental results show that the proposed approach can outperform classic and other deep neural network based anomaly detection methods. Our future work includes further theoretical study of the model and broadening the applications.

Appendix

A NETWORK STRUCTURES FOR IMAGE DATA SETS

The generator, discriminator, and the inference network for MNIST, CIFAR10 are given in Tables 5, 6, and 7. In the experiments, ImageNet32 use the same network structure as CIFAR10, and it is not stated in the tables.

Table 5: Generator of InvGAN in likelihood and anomaly detection experiments for MNIST and CIFAR10

<table>
<thead>
<tr>
<th>Generator</th>
<th>Layer</th>
<th>Number of Output</th>
<th>Kernel</th>
<th>Stride</th>
<th>Activation function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input (z \sim N(0, 1)^{2d})</td>
<td>50</td>
<td>50</td>
<td>Fully-Connected</td>
<td>128<em>4</em>4</td>
<td>ReLU</td>
</tr>
<tr>
<td>Transposed convolution</td>
<td>128<em>8</em>8</td>
<td>4*4</td>
<td>2</td>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Transposed convolution</td>
<td>64<em>16</em>16</td>
<td>4*4</td>
<td>2</td>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Transposed convolution</td>
<td>1<em>32</em>32 (MNIST)</td>
<td>4*4</td>
<td>2</td>
<td>Tanh</td>
<td></td>
</tr>
<tr>
<td>Transposed convolution</td>
<td>3<em>32</em>32 (CIFAR10)</td>
<td>4*4</td>
<td>2</td>
<td>Tanh</td>
<td></td>
</tr>
</tbody>
</table>

Table 6: Discriminator of InvGAN in likelihood and anomaly detection experiments for MNIST and CIFAR10

<table>
<thead>
<tr>
<th>Discriminator</th>
<th>Layer</th>
<th>Number of Output</th>
<th>Kernel</th>
<th>Stride</th>
<th>Activation function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input (x)</td>
<td>1<em>32</em>32 (MNIST)</td>
<td>4*4</td>
<td>2</td>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>64<em>16</em>16</td>
<td>4*4</td>
<td>2</td>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>128<em>8</em>8</td>
<td>4*4</td>
<td>2</td>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>128<em>4</em>4</td>
<td>4*4</td>
<td>2</td>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Fully-Connected</td>
<td>1</td>
<td>1</td>
<td>Sigmoid</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7: Inference network of InvGAN in likelihood and anomaly detection experiments for MNIST and CIFAR10

<table>
<thead>
<tr>
<th>Inference Network</th>
<th>Layer</th>
<th>Number of Output</th>
<th>Kernel</th>
<th>Stride</th>
<th>Activation function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input (x)</td>
<td>1<em>32</em>32 (MNIST)</td>
<td>4*4</td>
<td>2</td>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>64<em>16</em>16</td>
<td>4*4</td>
<td>2</td>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>128<em>8</em>8</td>
<td>4*4</td>
<td>2</td>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>128<em>4</em>4</td>
<td>4*4</td>
<td>2</td>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Fully-Connected</td>
<td>50</td>
<td>50</td>
<td>Sigmoid</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

B NETWORK STRUCTURES FOR ARRHYTHMIA DATA SET

The network structures for the generator, the discriminator, and the inference network in the Arrhythmia experiments are given in Tables 8, 9, and 10, respectively.

Table 8: Generator of InvGAN in Arrhythmia anomaly detection experiments

<table>
<thead>
<tr>
<th>Generator</th>
<th>Layer</th>
<th>Number of Output</th>
<th>Batch Normalization</th>
<th>Activation function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input (z \sim N(0, 1)^{2d,m})</td>
<td>50</td>
<td>50</td>
<td>Fully-Connected</td>
<td>128</td>
</tr>
<tr>
<td>Fully-Connected</td>
<td>256</td>
<td>256</td>
<td>Fully-Connected</td>
<td>128</td>
</tr>
<tr>
<td>Fully-Connected</td>
<td>274</td>
<td>274</td>
<td>Fully-Connected</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 9: Discriminator of InvGAN in Arrhythmia anomaly detection experiments

<table>
<thead>
<tr>
<th>Discriminator</th>
<th>Layer</th>
<th>Number of Output</th>
<th>Batch Normalization</th>
<th>Activation function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input (x)</td>
<td>274</td>
<td>274</td>
<td>Fully-Connected</td>
<td>256</td>
</tr>
<tr>
<td>Fully-Connected</td>
<td>128</td>
<td>128</td>
<td>Fully-Connected</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 10: Inference network of InvGAN in Arrhythmia anomaly detection experiments

<table>
<thead>
<tr>
<th>Inference Network</th>
<th>Layer</th>
<th>Number of Output</th>
<th>Batch Normalization</th>
<th>Activation function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input (x)</td>
<td>274</td>
<td>274</td>
<td>Fully-Connected</td>
<td>256</td>
</tr>
<tr>
<td>Fully-Connected</td>
<td>128</td>
<td>128</td>
<td>Fully-Connected</td>
<td>50</td>
</tr>
</tbody>
</table>