Abstract

A new method for outlier detection and generation is introduced by lifting data into the space of probability distributions which are not analytically expressible, but from which samples can be drawn using a neural generator. Given a mixture of unknown latent inlier and outlier distributions, a Wasserstein double autoencoder is used to both detect and generate inliers and outliers. The proposed method, named WALDO (Wasserstein Autoencoder for Learning the Distribution of Outliers), is evaluated on classical data sets including MNIST, CIFAR10 and KDD99 for detection accuracy and robustness. We give an example of outlier detection on a real retail sales data set and an example of outlier generation for simulating intrusion attacks. However we foresee many application scenarios where WALDO can be used. To the best of our knowledge this is the first work that studies both outlier detection and generation together.

Key Words: Outlier Detection, Outlier Generation, Wasserstein Distance, Wasserstein Autoencoder

1 Introduction

A well known definition of outliers states, “An outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism [2].” Many methods in outlier detection have been inspired by focusing on the deviation aspect of above definition.

For example, distance-based techniques define outliers as those data points that are far away from their neighbors; density-based approaches search for outliers in regions of low relative density; the one-class svm method defines outliers as those points that lie outside the tightest hypersphere containing most of the points [5, 1].

In this work we will focus both on the detection and generation mechanisms of outliers. In particular we will assume that data is generated from an unknown and unlabeled mixture of inlier and outlier distributions. We will have access to samples from only the unlabeled and the inlier distributions. Our primary objective will be to infer the outlier distribution without having recourse to outlier samples.

To infer the outlier distribution we will take a probabilistic view of autoencoders which have been used before for outlier detection [25]. An autoencoder can be seen as a self-mapping from an input space to itself mediated through a bottleneck - a lower dimensional representation of the data being mapped. In classical autoencoders, outliers are defined as those data points which have high reconstruction errors. A probabilistic view is to perceive the self-mapping as one inducing a new probability distribution on the input space, i.e., an autoencoder maps the original data distribution into a new distribution constrained by the bottleneck.

To compare probability distribution we use the Wasserstein distance as an alternate to the standard
Kullback-Liebler (KL) divergence \cite{21}. Autoencoders based on the Wasserstein distance (WAEs) have been recently proposed and shown to be accurate and efficient in generating complex distributions \cite{20, 15}. To generate outlier distributions, we leverage a recent approach to use a double decoder architecture to distinguish between inliers and outliers \cite{19}. An inlier and an outlier decoder (with a common encoder) compete with each other for data points based on reconstruction error and thus can be identified without setting a threshold parameter.

Our approach, WALDO, shown in Figure 1(a), encapsulates the double decoder framework using the Wasserstein distance, resulting in a generative detection model. For detection, the predicted class of a sample is given by the decoder with the least reconstruction error. For generation, a random sample in the latent space results in a generated inlier from the inlier decoder and in a generated outlier in the outlier decoder. Figure 1(b) shows how an inlier image in input (inliers are faces without glasses) is left unperturbed by the inlier decoder but the outlier decoder adds glasses to the face. Similarly an outlier image (face with glasses) is not changed by the outlier decoder but the inlier decoder removes the glasses. Similar transformations can be obtained indirectly by GANs but in a more complicated manner, by first creating a mean deviation vector from the decoder and then feeding it back through the generator \cite{16}.

The rest of the paper is structured as follows. In Section 2 we review related work with focus on deep learning models for outlier detection. We provide a short self-contained introduction to Wasserstein distance in Section 3 as that is a key building block of our approach. In Section 4, the WALDO autoencoder architecture is introduced consisting of four distinct components, together with the algorithm to train the model. A short theoretical analysis of our approach is the focus of Section 5. The experimental set up and results is the subject of Section 6. We conclude with a short discussion and
directions for future work in Section 7.

2 Related Work

Outlier detection is an extensively studied topic with diverse applications [1, 5]. With the advent of deep learning, variational auto-encoders (VAEs), generative adversarial networks (GANs) and other methods have been proposed for outlier detection [1]. While there are many recent works on deep learning models for outlier detection ([3], [14]), we will primarily survey the robust and generative ones in particular, as that is the focus of the paper.

Robust Methods: A customization of autoencoders for outlier detection is to make them robust, i.e., the model is not disproportionately affected by the presence of outliers. This in turn makes it easier to detect outliers as they will tend to have a higher reconstruction error. For example, Zhou et. al. [25] propose an autoencoder which decomposes the input data $X$ as sum of a low-dimensional manifold $U$ plus a sparse component $S$. A robust version of VAEs was recently proposed by Akrami et. al. [2] by using $\beta$-divergence instead of KL divergence as a loss function.

Generative Adversarial Networks (GANs) have been recently extended for anomaly detection [6]. Like variational autoencoders, GANs can map a random distribution (e.g., Gaussian) to an arbitrary data generating distribution $P_X$, through the use of a discriminator. Thus if we have a sample of “normal” data points then, in principle, we can learn the normal data manifold. However, BiGANs also learn an encoder function $E$ with the property that $E = G^{-1}$, where $G$ is the generator. Now given a query point $x$, if the reconstruction error $\|x - G(E(x))\|$ is large then $x$ is likely to be an outlier. In the experiment section, we will use one representative BiGAN as a baseline against which we will compare our proposed approach [24].

Threshold-Free Models: After a model has been built, outliers can either be identified based on ranking or thresholding. For example, if reconstruction error (RE) is used as a measure of outlierness, then data points can be either ranked based on the RE score or a threshold $\tau$ can be used such that those points whose $RE > \tau$, are labeled as outliers. Tian et. al. recently [19], proposed an autoencoder (CoRA) which uses two decoders: one for inliers and the other for outliers. Data points whose RE error is lower for the inlier decoder compared to the outlier decoder, were labeled as inliers and vice-versa. The use of two decoders frees the system from setting a pre-defined threshold value to identify outliers. We will use the idea of two decoders, in conjunction with the Wasserstein distance, to design an inlier and an outlier generative model.

3 Wasserstein Distance

Wasserstein Distance is a measure of dissimilarity between two probability distributions just like the KL divergence. Intuitively, Wasserstein Distance measures the amount of work required to move and transform one pile of sand to another and that is why a special case of it is referred to as the Earth Movers distance [21, 13].

While there are several equivalent ways to define Wasserstein Distance, we will use what is sometimes called as the probabilistic definition. The p-Wasserstein distance $W_p$ between a probability measure $\mu_1$ and $\mu_2$ on $\mathbb{R}^d$ is defined as

$$W_p(\mu_1, \mu_2) = \inf_{X \sim \mu_1, Y \sim \mu_2} \mathbb{E} \|X - Y\|^{1/p}$$
3.1 Wasserstein Autoencoders (WAEs)

An autoencoder based on \( W_p \) (WAE) was recently proposed [20]. Consider an autoencoder \( h : X \xrightarrow{Q} Z \xrightarrow{G} X \). Let \( P_X \) be the original distribution and \( h\#P_X \) be the output distribution induced by \( h \). Then a WAE learns a function \( h \) which minimizes \( W_p(P_X, h\#P_X) \). However, both the encoding (\( Q \)) and decoding function (\( G \)) can be viewed in a probabilistic fashion. Thus, if \( Q \) and \( P \) are coupled and expressed in terms of \( P \) and \( W \), then a WAE learns a function \( Z \) which minimizes \( W_p(P_X, h\#P_X) \).

3.1.1 Constraints: We do not have access to samples from \( P_X \).

4 Problem Definition and WALDO

We now formally define the problem and propose the WALDO architecture as a solution.

**Given:** \( P_X = (1 - \nu)P_X^i + \nu P_X^o \) be a mixture of an inlier and an outlier distribution on an input space \( X = \mathbb{R}^d \) for \( 0 < \nu < 1 \). No assumptions are made on the analytical form of the three distributions: \( P_X^i, P_X^o \) and \( P_X^\gamma \). Let \( X^u \) and \( X^p \) be samples from \( P_X^u \) and \( P_X^p \).

**Objective:** Learn generating distributions \( P_G \) and \( P_G^o \) on \( X \) which minimize \( W_p(P_X^i, P_G^i) \) and \( W_p(P_X^p, P_G^o) \).

4.1 WALDO Architecture

The architecture of WALDO is a generalization of the WAE [20] and CoRa [19] to simultaneously detect and generate inliers and outliers. WALDO consists of four components as shown in Figure 1.

1. An inlier decoder and generator denoted as \( G^i \) which maps the latent space \( Z \) to the output space \( X \). The inlier decoder will induce a distribution \( P_G^i \). Once trained, \( G^i \) can take an element generated from \( P_Z \) and produce samples which will appear to be from \( P_X^i \).

2. An outlier decoder and generator denoted as \( G^o \). Like the inlier decoder, the outlier decoder can be used to generate outlier samples which will appear to be from \( P_X^o \).

3. A common encoder \( Q \) which maps the input space \( X \) into the latent space \( Z \). In the original WAE paper, the constraint \( \mathbb{E}_{X \sim P_X}(Q|Z) = P_Z \) is enforced using an adversarial discriminator loss. However in WALDO we have the option of either enforcing \( \mathbb{E}_{X \sim P_X^i}(Q|Z) = P_Z \) or \( \mathbb{E}_{X \sim P_X^o}(Q|Z) = P_Z \). In our experiments we have consistently observed that the former gave better results than the latter. This is not unexpected as by only enforcing the constraints on the inliers there will be a smaller chance that the inliers and the outliers will be mapped to the same region of the latent space.

4. A discriminator \( D_\gamma \) trained in an adversarial manner like in traditional GANs. The role of \( D_\gamma \) is to enforce the constraint \( Q_Z = P_Z \). However, unlike traditional GANs, \( D_\gamma \) operates in the lower-dimensional latent space \( Z \). Recall in adversarial learning, the encoder \( Q \) is trying to “fool” the discriminator to treat its samples as those from the prior \( P_Z \).
Algorithm 1: Wasserstein Autoencoder for Learning Distribution of Outliers (WALDO)

**Input:** positive data $X^p$, unlabeled test data $X^u$

1. Initialize the parameters of the encoder $Q_\phi$, inlier decoder $G_i^\theta$, discriminator $D_\gamma$. Set the outlier decoder $G_o^\omega$ parameter $\omega = \theta$.

2. **while** $\phi, \theta, \omega$ not converged **do**
   3. Sample batch of size $n$ from $X^p$ positive and $X^u$ unlabeled data, $|X^p \cup X^u| = n$
   4. Sample $\{z_1, \ldots, z_n\}$ from the prior $P_Z$
   5. Sample $\{\hat{z}_1, \ldots, \hat{z}_n\}$ from $Q_\theta(x)$ for $i = 1, \ldots, n$
   6. Update $D_\gamma$ by ascending:
      $$\frac{1}{n} \sum_{j=1}^{n} \log D_\gamma(z_j) + \log(1 - D_\gamma(\hat{z}_j))$$
      Compute advantage of inlier decoder:
      $$adv^i = \min_{\forall x_i \in X^u} ||G_o^\omega(\hat{z}_i) - x_j||_2^2 - \min_{\forall x_j \in X^p} ||G_i^\theta(\hat{z}_j) - x_j||_2^2$$
   7. **for** $j = 1, \ldots, n$ **do**
      8. **if** $||G_i^\theta(\hat{z}_j) - x_j||_2^2 + adv^i < ||G_o^\omega(\hat{z}_j) - x_j||_2^2 \forall x_j \in X^p$ **then**
      9. $y_j = 0$
     10. **else**
     11. $y_j = 1$
     12. **end**
   13. **end**
   14. Update $Q_\phi, G_i^\theta$ and $G_o^\omega$ by descending:
      $$\frac{1}{n} \sum_{j=1}^{n} (y_j||G_o^\omega(\hat{z}_i) - x_i||_2^2 + (1 - y_j)||G_i^\theta(\hat{z}_j) - x_j||_2^2) - \lambda \cdot (\log(D_\gamma(\hat{z}_j)))$$

4.2 **Algorithm**

WALDO is defined in Algorithm 1. First the discriminator is trained by ascending (line 7) to discriminate between samples from the prior $P_z$ and samples from the encoder $Q_\theta$. In practice only encoded inliers samples will be forced to match the prior distribution (positive-only $D_\gamma$ training). In the training of the autoencoder, only the decoder with lower reconstruction error will be selected in the loss for each data point (lines 9-16). Note that in the competition for a data point, the inlier decoder has seen more samples, thus it has a natural advantage over the outlier decoder in decoding both outliers and inliers. Conversely but less frequently, a random initialization may lead to an advantage of the outlier decoder, with a consequently spurious training during the initial epochs. To cope with both cases of imbalance we introduce an advantage term (line 8), that penalizes the reconstruction error of the decoder with the best reconstruction error.
5 Analysis of WALDO

We analyze theoretical aspects of WALDO for the special case of $p = 2$. In particular, we show that under certain circumstances, $W_2(P_X^a, P_G^a)$ upper bounds a positive weighted sum of $W_2(P_X^c, P_G^c)$, $W_2(P_X^i, P_G^i)$ and $W_2(Q_Z, P_Z)$. Thus by minimizing an upper bound we can indirectly optimize the decoders. We use the following characterization of WAE [15] for decoders with the added assumption that they are Lipschitz with constant $\gamma$.

$$W_2(P_X, P_G) = \inf_Q \sqrt{\mathbb{E}_{X \sim P_X} \| X - G(Q(X)) \|^2} + \gamma W_2(Q_Z, P_Z)$$

**Theorem 1.** For a system with an inlier decoder $P_G^i$ and an outlier decoder $P_G^o$ and a shared deterministic encoder $Q$, the following holds:

$$W_2(P_X^a, P_G^a) \geq \sqrt{\frac{\nu}{2}} W_2(P_X^c, P_G^c) + \sqrt{\frac{1-\nu}{2}} W_2(P_X^i, P_G^i) + \gamma \left( 1 - \sqrt{\frac{\nu}{2}} - \sqrt{\frac{1-\nu}{2}} \right) W_2(Q_Z, P_Z)$$

**Proof.** See Supplementary Text.

**Implication of Theorem:** The above theorem shows that by using the Wasserstein metric we can formally distribute the error between the unlabeled data distribution $P_X^a$ and the outlier (inlier) generator $P_G^o$ across the two two decoders. The coefficients $\sqrt{\frac{\nu}{2}}$ also suggests that if $\nu$ is very small then an algorithm which tries to minimize $W_2(P_X^a, P_G^c)$ will effectively expend “more effort” in optimizing $W_2(P_X^i, P_G^i)$ than $W_2(P_X^c, P_G^c)$). The use of Advantage in the algorithm is way to compensate the natural weakness of optimizing the outlier decoder due to the small value of $\nu$ even though the dependence is improved as the factor is $\sqrt{\frac{\nu}{2}}$ will be higher than $\nu < 1/2$. Note that in practice the Lipschitz condition can be enforced using gradient clipping.

6 Experiments

In this section we empirically evaluate the effectiveness of WALDO. We report on four sets of experiments.

1. We carry out an ablation study of WALDO by varying its internal components. Specifically, we evaluate the accuracy of WALDO when the discriminator is applied to only inlier data, i.e., data sampled from $P_X^i$. Similarly the impact of training WALDO with and without the use of advantage is tested.

2. We evaluate WALDO on its ability to generate outliers. We test whether $P_G^o$ can be used to generate new network intrusion attacks using the KDD99 data set.

3. We present a real case study where WALDO is applied on real sales data to accurately discover extremely rare patterns with high recall.

4. Finally we compare WALDO with other state of the art and representative deep learning based methods for anomaly detection: DeepSVDD [18]. ALAD [23], WAE [20] and CoRA [19]. The comparison is carried out by varying contamination level of the training set and outlier ratios in test dataset.

6.1 Datasets

We use four publicly available datasets for experiments:

1. **MNIST** [10]: containing 60k training samples and 10k test samples from 10 digit classes. Each digit is a $28 \times 28$ grayscale image. We choose the digit 0 as the inlier class and the others as outliers.

2. **Fashion MNIST** [22]: consisting of 60k training samples and 10k test samples from 10 classes. Each sample is a $28 \times 28$ grayscale image in a clothes category. We use the class 0 as inliers ($X^i$), and the others as outliers.

3. **KDD99** [12]: a large-scale network traffic data with 121 features in each sample. We use 10% of the dataset to extract the inliers ($X^i$) and another 10% for the unlabeled data. This
data set is also used to show the capability of WALDO to generate new meaningful attacks.

4. CIFAR10 [9]: consisting of 60k $32 \times 32$ color images in 10 classes including 50k training and 10k test images.

6.2 Ablation study.

**Impact of Positive-only $D_\gamma$ on WALDO.** Training the discriminator $D_\gamma$ only on the positive data (labeled inliers) helps the model in separating the two distributions in the latent space, having effects both on the latent space and on the output space. In Figure 2a we aggregate the effects of positive training for $D_\gamma$, showing the distance of the encoded samples from the mean of the distribution $P_Z$. As expected, the outliers get mapped further from the mean of the prior distribution $P_Z$, while the inliers are closer.

**Impact of Advantage on WALDO.** Recall that in WALDO the two decoders, $G^i$ and $G^o$ compete with each other to get points assigned to them. However because of the availability of the $X^i$ set, $G^i$ has a natural advantage to have a low reconstruction error on data points in $X^u$. To overcome the natural bias, we introduced the Advantage term (see Line 9 in Algorithm 1). In Figure 2b we show how using the advantage penalty substantially improves the results (higher F1 score), in particular when the outlier ratio becomes smaller. Moreover, we observed how employing the advantage penalty resulted in more reproducible results over different random seeds, as can be seen from the smaller variance in the accuracy.

6.3 Extreme Outlier Discovery: Real Case Study

We give an example of how WALDO can be used to detect extremely rare patterns on a real data set acquired from a large retailer. We look at the weekly sales pattern of one product $X$ over nearly four years, (from 2016/03/20 to 2019/12/29 in all 52 price-markets in the US, pre COVID-19), which is typically sold more on weekends than weekdays. Each data point is a vector of seven dimensions, and we took a small fraction of the data points and labeled them as inliers if the volume of the product sold on either Saturday or Sunday was greater than any of the weekdays. There was a total of 10,234 inliers and 62 outliers. Thus the percentage of outliers was 0.61%. WALDO was only given a small labeled sample of inliers (2,047) and all the remaining set was unlabeled. Note we chose this pattern, which is “easy to query” as it makes it straightforward to characterize outliers. Recall WALDO does not see any labeled outliers. Here are the key observations:

1. All the outliers were assigned to the outlier decoder (high recall).
2. However, many inliers were also assigned to the outlier decoder just because of the extreme skewness of the data set (low precision).
3. If we ranked all the data points by reconstruction error of the inlier decoder we observed an average precision (AP) of close to 46% for the outliers. Thus even though outliers constitute an extremely small percentage of the data, we are able to locate them at the top of the list. This demonstrates that WALDO has the promise to detect rare patterns.

![Figure 3: Outlier detection of retail patterns. Outliers constitute a very small percentage of the data.](image)

6.4 Capability of generating new attacks on the KDD99 Data Set

One of the novelties of our work is to provide the capability of generating new outliers. For example we can generate realistic new attacks using the
KDD99 dataset. Our attacks are generated using the trained WALDO network architecture for the data set. Both positive and unlabeled were used in the training model. We used trained encoder on the inliers and outliers independently from KDD99 to generate the encoded data in which two distributions are formed. From the two distributions, we sampled two independent groups of Gaussian random variables

\[ X_i \sim \mathcal{N}(P_{Z_i}, \sigma^2_i), \quad X_o \sim \mathcal{N}(P_{Z_o}, \sigma^2_o) \].

Then we decoded the samples to reconstruct the data.

6.5 Convergence Efficiency

In this experiment, we measure the performance of WALDO during training by tracking the number of epochs needed to converge to a stable result on MNIST. In Figure 6, the results of WALDO is compared to CoRa [19], both in its original setting and improved with the proposed Advantage penalty, and the Wasserstein Autoencoder (WAE [20], showing the average precision (AP) mean and standard deviation over eight runs on the unlabeled set. The results show that WALDO reaches higher precision faster, while convergence is obtained consistently with very small variance across different runs.
Table 1: Evaluation and comparison with state of the art models, showing average values of multiple runs for AUC and AUPRC. In **bold**: best result for selected outlier ratio and collection.

6.6 WALDO vs. other methods

We compare the detection accuracy of WALDO with other methods using AUC and AUPRC metrics. At the outset we caution that direct comparison between the methods is problematic because of the nature of the methods. For example, DeepSVDD [17], ALAD [24] and even WAE [20] require a threshold for determining outliers while both CoRa [19] and WALDO do not. For example, there are two versions of DeepSVDD, the first called **soft boundary Deep SVDD** requires a hyperparameter that controls the trade-off on how many data points are allowed to fall outside the hypersphere boundary. The second version scores each point based on the distance from the center of the hypersphere induced - the further the distance the more likely it is an outlier and thus requires a threshold cutoff to label points as outliers. Also we have improved CoRa [19] with **Advantage** to make it more stable and we compare against this improved version.

The comparison between the methods is shown in Table 1. The $\nu$ column represents the percentage of outliers in the mixture. In thirteen out of the sixteen cases, WALDO does better on the AUPRC metric and in ten out of sixteen it does better on AUC. On MNIST, both WAE and CoRa have wide confidence intervals while those of WALDO are relatively tight. On Fashion MNIST at $\nu = 0.1$, DeepSVDD
AUPRC tends to be lower than AUC across the meth-  

Table 2: Robustness Evaluation and comparison with state of the art models, showing AUC and AUPRC.

| Cont. | DeepSVDD \[18\] | ALAD \[24\] | WAE \[29\] | CoRa \[19\] + Adr. | WALDO |
|-------|----------------|----------------|------|----------------|-------|
| \(\nu\) | AUC | AUPRC | AUC | AUPRC | AUC | AUPRC | AUC | AUPRC | AUC | AUPRC | AUC | AUPRC |
| 0.05 | 96.73 | 66.94 | 66.25 | 67.11 | 97.52 | 62.43 | 91.38 | 23.11 | 97.82 | 63.43 |
| \(5\%\) | 0.91 | 46.45 | 74.27 | 72.09 | 17.63 | 94.41 | 67.82 | 93.14 | 41.17 | 98.32 | 86.05 |
| 0.2 | 95.87 | 83.69 | 56.60 | 20.39 | 85.23 | 47.28 | 94.73 | 69.36 | 99.68 | 98.68 |
| 0.5 | 93.77 | 92.80 | 50.41 | 47.04 | 69.01 | 70.99 | 98.18 | 97.89 | 99.51 | 99.56 |

| \(15\%\) | 0.95 | 89.68 | 83.58 | 16.65 | 94.83 | 23.86 | 88.99 | 21.15 | 93.36 | 36.02 |
| \(10\%\) | 0.95 | 83.91 | 72.33 | 17.63 | 90.28 | 33.43 | 90.63 | 40.34 | 97.24 | 82.96 |
| 0.5 | 94.95 | 83.69 | 79.65 | 42.49 | 82.98 | 54.10 | 92.63 | 61.46 | 99.12 | 97.10 |
| 0.9 | 93.43 | 83.09 | 81.44 | 77.11 | 88.40 | 86.53 | 98.18 | 87.89 | 99.51 | 99.56 |

| | 0.05 | 83.66 | 21.31 | 35.66 | 64.96 | 88.22 | 22.56 | 87.85 | 44.41 | 88.55 | 41.18 |
| \(5\%\) | 0.91 | 84.92 | 33.86 | 38.83 | 12.63 | 88.62 | 39.79 | 87.82 | 62.54 | 90.21 | 66.22 |
| 0.2 | 83.72 | 49.30 | 39.73 | 21.83 | 89.04 | 62.36 | 86.40 | 72.55 | 89.71 | 75.57 |
| 0.5 | 81.07 | 77.66 | 39.92 | 48.13 | 88.67 | 85.44 | 84.62 | 87.47 | 87.95 | 88.92 |

| FMNIST | 0.05 | 83.89 | 32.98 | 35.57 | 0.94 | 84.22 | 17.88 | 87.85 | 44.53 | 88.24 | 43.10 |
| \(1\%\) | 0.91 | 83.68 | 36.72 | 39.78 | 12.57 | 83.23 | 29.92 | 87.82 | 62.32 | 90.15 | 66.30 |
| 0.2 | 82.56 | 43.98 | 39.71 | 21.83 | 85.03 | 48.22 | 86.52 | 72.54 | 89.53 | 75.80 |
| 0.5 | 82.05 | 71.32 | 39.91 | 48.11 | 84.87 | 80.80 | 84.63 | 87.49 | 87.54 | 88.81 |

| CIFAR10 | 0.05 | 68.89 | 6.42 | 77.14 | 12.12 | 78.88 | 13.15 | 78.91 | 13.12 | 79.04 | 13.19 |
| \(5\%\) | 0.91 | 70.66 | 11.92 | 71.94 | 21.03 | 74.03 | 23.20 | 74.09 | 23.20 | 74.09 | 23.23 |
| 0.2 | 51.78 | 20.77 | 68.27 | 33.24 | 70.56 | 35.39 | 70.47 | 35.29 | 70.59 | 35.37 |
| 0.5 | 51.40 | 50.23 | 67.78 | 63.66 | 71.45 | 67.44 | 71.19 | 67.31 | 71.35 | 67.44 |

| KDD | 0.05 | 95.18 | 37.19 | 99.04 | 75.85 | 96.94 | 41.20 | 97.20 | 45.20 | 99.44 | 79.89 |
| \(1\%\) | 0.91 | 95.29 | 55.53 | 99.13 | 86.34 | 96.78 | 56.87 | 97.41 | 64.82 | 99.12 | 83.54 |
| 0.2 | 95.30 | 73.70 | 99.18 | 93.48 | 96.43 | 71.30 | 97.40 | 79.92 | 99.44 | 95.22 |
| 0.5 | 96.06 | 91.71 | 98.74 | 91.90 | 94.88 | 87.44 | 97.45 | 94.66 | 99.16 | 98.23 |

| \(10\%\) | 0.91 | 93.34 | 29.47 | 99.31 | 78.79 | 95.24 | 31.82 | 97.22 | 45.17 | 97.71 | 50.65 |
| 0.2 | 94.28 | 67.52 | 98.61 | 91.30 | 94.61 | 63.71 | 97.05 | 77.17 | 98.69 | 87.74 |
| 0.5 | 94.74 | 89.37 | 86.79 | 70.65 | 92.82 | 85.42 | 96.91 | 92.62 | 98.44 | 96.52 |

Table 2: Robustness Evaluation and comparison with state of the art models, showing AUC and AUPRC. The \textit{Cont.} column represents the percentage of outliers in the positive training data and \(\nu\) column represents the percentage of outliers in the mixture. In \textbf{bold}: best result for selected \(\nu\), Cont. and collection.

has a slightly higher AUPRC but \textit{WALDO} has a tighter confidence interval. In summary we can conclude that \textit{WALDO} is competitive outlier detection approach vis-a-vis representative deep learning methods. An observation worth highlighting is that AUPRC tends to be lower than AUC across the methods. This suggests that there is lot of room for improvement for outlier detection methods in general.

### 7 Discussion and Conclusion

We propose \textit{WALDO}, an extension of deep autoencoders to both detect and generate outliers. \textit{WALDO} uses the Wasserstein metric distance (WD) to train an autoencoder which has an inlier and an outlier decoder but a common encoder. The WD is ideally suitable for outlier detection as it can gracefully handle distributions which may not have identical support. We give an example of detecting extremely rare patterns on a retailer data set. Besides being an accurate outlier detector, \textit{WALDO} can be used to generate outliers. This may have potentially widespread application including rare event simulation and data augmentation. We give one example where outlier generation can be used to create network attacks using the benchmark KDD99 Cup data set. Applying \textit{WALDO} to application areas like health, transportation and climate change, where outliers are ever present, may yield promising insights.
References

[1] Charu C Aggarwal. *Outlier Analysis*. Springer, 2nd edition, 2016.
[2] Haleh Akrami, Anand A. Joshi, Jian Li, Sergul Aydore, and Richard M. Leahy. Robust variational autoencoder, 2019.
[3] Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407*, 2019.
[4] Raghavendra Chalapathy, Aditya Krishna Menon, and Sanjay Chawla. Robust, deep and inductive anomaly detection. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 36–51. Springer, 2017.
[5] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Outlier detection: A survey. *ACM Computing Surveys*, 2007.
[6] Federico Di Mattia, Paolo Galeone, Michele De Simoni, and Emanuele Ghelli. A survey on gans for anomaly detection. *arXiv preprint arXiv:1906.11632*, 2019.
[7] Peter J Huber. *Robust statistics*, volume 523. John Wiley & Sons, 2004.
[8] Ryuichi Kiryo, Gang Niu, Marthinus C du Plessis, and Massashi Sugiyama. Positive-unlabeled learning with non-negative risk estimator. In *Advances in neural information processing systems*, pages 1675–1685, 2017.
[9] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).
[10] Yann LeCun. The mnist database of handwritten digits, 1998.
[11] Xiao-Li Li and Bing Liu. Learning from positive and unlabeled examples with different data distributions. In *European conference on machine learning*, pages 218–229. Springer, 2005.
[12] Moshe Lichman. Uci machine learning repository. Irvine, ca: University of california, school of information and computer science, 1998.
[13] Victor M. Panaretos and Yoav Zemel. Statistical aspects of wasserstein distances. *Annual Review of Statistics and Its Application*, 6(1):405–431, 2019.
[14] Guansong Pang, Chunhua Shen, and Anton van den Hengel. Deep anomaly detection with deviation networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD ’19, page 353–362, 2019.
[15] Giorgio Patrini, Rianne van den Berg, Patrick Forre, Marcello Carioni, Samarth Bhargav, Max Welling, Tim Ge newein, and Frank Nielsen. Sinkhorn autoencoders. *arXiv preprint arXiv:1810.01118*, 2018.
[16] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016*, 2016.
[17] Markus Ring, Daniel Schlör, Dieter Landes, and Andreas Hotho. Flow-based network traffic generation using generative adversarial networks. *arXiv preprint arXiv:1810.07795*, 2018.
[18] Lukas Ruff, Nico Görnitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Robert Vandermeulen, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In *International Conference on Machine Learning*, pages 4390–4399, 2018.
[19] Kai Tian, Shuigeng Zhou, Jianping Fan, and Jihong Guan. Learning competitive and discriminative reconstructions for anomaly detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5167–5174, 2019.
[20] I Tolstikhin, O Bousquet, S Gelly, and B Schölkopf. Wasserstein auto-encoders. In *International Conference on Learning Representations (ICLR 2018)*, 2018.
[21] Cedric Villani. *Topics in Optimal Transportation*. AMS, 2013.
[22] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.
[23] Houssam Zenati, Chuan Sheng Foo, Bruno Lecouat, Gaurav Manek, and Vijay Ramaseshan Chandrasekhar. Efficient gan-based anomaly detection. *arXiv preprint arXiv:1802.06222*, 2018.
[24] Houssam Zenati, Manon Romain, Chuan-Sheng Foo, Bruno Lecouat, and Vijay Chandrasekhar. Adversarially learned anomaly detection. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2018.
[25] Chong Zhou and Randy C Paffenroth. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 665–674. ACM, 2017.
**Analysis of WALDO**

We analyze theoretical aspects of WALDO for the special case of $p = 2$. In particular, we show that under certain circumstances, $W_2(P_X, P_G)$ upper bounds a positive weighted sum of $W_2(P_X^u, P_G^o)$, $W_2(P_X^i, P_G^i)$ and $W_2(Q_Z, P_Z)$. Thus by minimizing an upper bound we can indirectly optimize the decoders. We use the following characterization of WAE [20] for decoders with the added assumption that they are Lipschitz with constant $\gamma$.

$$
W_2(P_X, P_G) = \inf_Q \sqrt{\mathbb{E}_{X \sim P_X} ||X - G(Q(X))||^2} + \gamma W_2(Q_Z, P_Z)
$$

**Theorem 2.** For a system with an inlier decoder $P_X^i$ and an outlier decoder $P_X^o$ and a shared deterministic encoder $Q$, there exists positive constants $\alpha, \beta, \delta$ such that

$$
W_2(P_X^u, P_G^o) \geq \alpha W_2(P_X^o, P_G^o) + \beta W_2(P_X^i, P_G^i) + \delta W_2(Q_Z, P_Z)
$$

**Proof.**

$$
W_2(P_X^u, P_G^o) = \inf_Q \sqrt{\mathbb{E}_{X \sim P_X} ||X - G^o(Q(X))||^2} + \gamma W_2(Q_Z, P_Z)
$$

$$
\geq \sqrt{\frac{\nu}{2}} \inf_Q \sqrt{\mathbb{E}_{X \sim P_X} ||X - G^o(Q(X))||^2} + \frac{1 - \nu}{2} \inf_Q \sqrt{\mathbb{E}_{X \sim P_X} ||X - G^o(Q(X))||^2} + \gamma W_2(Q_Z, P_Z)
$$

$$
\geq \sqrt{\frac{\nu}{2}} W_2(P_X^o, P_G^o) + \frac{1 - \nu}{2} W_2(P_X^i, P_G^i) + \\
\gamma \left(1 - \sqrt{\frac{\nu}{2}} - \sqrt{\frac{1 - \nu}{2}}\right) W_2(Q_Z, P_Z)
$$

The proof uses that (i) $\sqrt{a + b} \geq \frac{1}{\sqrt{2}}(\sqrt{a} + \sqrt{b})$ and (ii) $\inf(f(x) + g(x)) \geq \inf(f(x)) + \inf(g(x))$ and that, (iii) the reconstruction error of the optimal decoder $G^o(G^i)$ is the smallest on $P_X^o(P_X^i)$ and (iii) the network shares common encoder $Q_Z$. A similar bound holds for $W_2(P_X^u, P_G^i)$.

**Supplement for reproducibility**

All Python implementations of the methods evaluated in the paper can be found in Table 3. Hyperparameters settings vary for specific collections and methods and can be found in the source code of the implementations.
Table 3: Source code repositories for each method evaluated in this work.

| Method               | Source code of implementation | Framework |
|----------------------|------------------------------|-----------|
| DeepSVDD             | https://github.com/lukasruff/Deep-SVDD-PyTorch | PyTorch   |
| ALAD                 | https://github.com/houssamzenati/Adversarially-Learned-Anomaly-Detection | TensorFlow |
| Wasserstein AE       | https://github.com/sedelmeyer/wasserstein-auto-encoder/blob/master/Wasserstein-auto-encoder_tutorial.ipynb | PyTorch |
| CoRA (+ advantage)   | https://github.com/kddblind/waldo | PyTorch |
| WALDO                | https://github.com/kddblind/waldo | PyTorch |

7.1 Neural Network architectures

For WALDO, CoRa and WAE, we use autoencoders without any convolutional layers for KDD, and convolutional autoencoders for image datasets (Cifar-10, Fashion-MNIST, MNIST). A detailed table to show each layer for each datasets can be found in next page.

| Non Linearity | Output Shape | Parameters |
|---------------|--------------|------------|
| Encoder       |              |            |
| Linear-1      | [-1, 120, 64] | 7,808      |
| LeakyReLU-2   | [-1, 120, 64] | 0          |
| Linear-3      | [-1, 120, 32] | 2,080      |
| ReLU-4        | [-1, 120, 32] | 0          |
| Inlier Decoder|              |            |
| Linear-1      | [-1, 120, 64] | 2,112      |
| LeakyReLU-2   | [-1, 120, 64] | 0          |
| Linear-3      | [-1, 120, 121]| 7,865      |
| Tanh-4        | [-1, 120, 121]| 0          |
| Outlier Decoder|             |            |
| Linear-1      | [-1, 120, 64] | 2,112      |
| LeakyReLU-2   | [-1, 120, 64] | 0          |
| Linear-3      | [-1, 120, 121]| 7,865      |
| Tanh-4        | [-1, 120, 121]| 0          |
| Discriminator |              |            |
| Linear-1      | [-1, 120, 64] | 1,056      |
| LeakyReLU-2   | [-1, 120, 64] | 0          |
| Dropout-3     | [-1, 120, 64] | 0          |
| Linear-4      | [-1, 120, 1]  | 33         |
| Sigmoid-5     | [-1, 120, 1]  | 0          |

Table 4: KDD Architecture
| Non Linearity | Output Shape | Parameters |
|----------------|--------------|------------|
| Encoder        |              |            |
| Conv2d-1       | (-1, 32, 16, 16) | 1,536      |
| ReLU-2         | (-1, 32, 16, 16) | 0          |
| Conv2d-3       | (-1, 64, 8, 8)  | 32,768     |
| BatchNorm2d-4  | (-1, 64, 8, 8)  | 128        |
| ReLU-5         | (-1, 64, 8, 8)  | 0          |
| Conv2d-6       | (-1, 128, 4, 4) | 131,072    |
| BatchNorm2d-7  | (-1, 128, 4, 4) | 256        |
| ReLU-8         | (-1, 128, 4, 4) | 0          |
| Conv2d-9       | (-1, 256, 2, 2) | 524,288    |
| BatchNorm2d-10 | (-1, 256, 2, 2) | 512        |
| ReLU-11        | (-1, 256, 2, 2) | 0          |
| Linear-12      | (-1, 32)       | 32,800     |
| Inlier Decoder |              |            |
| Linear-1       | (-1, 3, 20736) | 684,288    |
| ReLU-2         | (-1, 3, 20736) | 0          |
| ConvTranspose2d-3 | (-1, 128, 12, 12) | 524,416  |
| BatchNorm2d-4  | (-1, 128, 12, 12) | 256       |
| ReLU-5         | (-1, 128, 12, 12) | 0         |
| ConvTranspose2d-6 | (-1, 64, 15, 15) | 131,136  |
| BatchNorm2d-7  | (-1, 64, 15, 15) | 128       |
| ReLU-8         | (-1, 64, 15, 15) | 0         |
| ConvTranspose2d-9 | (-1, 3, 32, 32) | 3075      |
| Sigmoid-10     | (-1, 3, 32, 32) | 0          |
| Outlier Decoder|              |            |
| Linear-1       | (-1, 3, 20736) | 684,288    |
| ReLU-2         | (-1, 3, 20736) | 0          |
| ConvTranspose2d-3 | (-1, 128, 12, 12) | 524,416  |
| BatchNorm2d-4  | (-1, 128, 12, 12) | 256       |
| ReLU-5         | (-1, 128, 12, 12) | 0         |
| ConvTranspose2d-6 | (-1, 64, 15, 15) | 131,136  |
| BatchNorm2d-7  | (-1, 64, 15, 15) | 128       |
| ReLU-8         | (-1, 64, 15, 15) | 0         |
| ConvTranspose2d-9 | (-1, 3, 32, 32) | 3075      |
| Sigmoid-10     | (-1, 3, 32, 32) | 0          |
| Discriminator  |              |            |
| Linear-1       | (-1, 3, 128)   | 4,224      |
| ReLU-2         | (-1, 3, 128)   | 0          |
| Linear-3       | (-1, 3, 128)   | 16,512     |
| ReLU-4         | (-1, 3, 128)   | 0          |
| Linear-5       | (-1, 3, 128)   | 16,512     |
| ReLU-6         | (-1, 3, 128)   | 0          |
| Linear-7       | (-1, 3, 128)   | 16,512     |
| ReLU-8         | (-1, 3, 128)   | 0          |
| Linear-9       | (-1, 3, 1)     | 129        |
| Sigmoid-10     | (-1, 3, 1)     | 0          |

Table 5: CIFAR10 Architecture
Table 6: MNIST & FMNIST Architecture

| Non Linearity | Output Shape | Parameters |
|---------------|--------------|------------|
| Encoder       |              |            |
| Conv2d-1      | (-1, 40, 14, 14) | 640        |
| Conv2d-3      | (-1, 80, 7, 7)  | 51,200     |
| BatchNorm2d-4 | (-1, 80, 7, 7)  | 160        |
| ReLU-5        | (-1, 80, 7, 7)  | 0          |
| Conv2d-6      | (-1, 160, 3, 3)| 204,800    |
| BatchNorm2d-7 | (-1, 160, 3, 3)| 320        |
| ReLU-8        | (-1, 160, 3, 3)| 0          |
| Conv2d-9      | (-1, 320, 1, 1)| 819,200    |
| BatchNorm2d-10| (-1, 320, 1, 1)| 640        |
| ReLU-11       | (-1, 320, 1, 1)| 0          |
| Linear-12     | (-1, 32)      | 10,272     |
| Inlier Decoder|              |            |
| Linear-1      | (-1, 3, 15680)| 517,440    |
| ReLU-2        | (-1, 3, 15680)| 0          |
| ConvTranspose2d-3 | (-1, 160, 10, 10) | 819,360 |
| BatchNorm2d-4 | (-1, 160, 10, 10)| 320    |
| ReLU-5        | (-1, 160, 10, 10)| 0      |
| ConvTranspose2d-6 | (-1, 80, 13, 13)| 204,880 |
| BatchNorm2d-7 | (-1, 80, 13, 13)| 160    |
| ReLU-8        | (-1, 80, 13, 13)| 0      |
| ConvTranspose2d-9 | (-1, 1, 28, 28) | 1,281   |
| Sigmoid-10    | (-1, 1, 28, 28)| 0      |
| Outlier Decoder|            |            |
| Linear-1      | (-1, 3, 15680)| 517,440    |
| ReLU-2        | (-1, 3, 15680)| 0          |
| ConvTranspose2d-3 | (-1, 160, 10, 10) | 819,360 |
| BatchNorm2d-4 | (-1, 160, 10, 10)| 320    |
| ReLU-5        | (-1, 160, 10, 10)| 0      |
| ConvTranspose2d-6 | (-1, 80, 13, 13)| 204,880 |
| BatchNorm2d-7 | (-1, 80, 13, 13)| 160    |
| ReLU-8        | (-1, 80, 13, 13)| 0      |
| ConvTranspose2d-9 | (-1, 1, 28, 28) | 1,281   |
| Sigmoid-10    | (-1, 1, 28, 28)| 0      |
| Discriminator |              |            |
| Linear-1      | (-1, 3, 160)  | 5,280      |
| ReLU-2        | (-1, 3, 160)  | 0          |
| Linear-3      | (-1, 3, 160)  | 25,760     |
| ReLU-4        | (-1, 3, 160)  | 0          |
| Linear-5      | (-1, 160, 10, 10)| 25,760 |
| ReLU-6        | (-1, 3, 160)  | 0          |
| Linear-7      | (-1, 3, 160)  | 25,760     |
| ReLU-8        | (-1, 3, 160)  | 0          |
| Linear-9      | (-1, 3, 1)    | 161        |
| Sigmoid-10    | (-1, 3, 1)    | 0          |