AMAD: Adversarial Multiscale Anomaly Detection on High-Dimensional and Time-Evolving Categorical Data

Zheng Gao  
Indiana University Bloomington  
gao27@indiana.edu

Lin Guo  
Alibaba Group  
lin.gl@alibaba-inc.com

Chi Ma  
Alibaba Group  
beiji.mc@alibaba-inc.com

Xiao Ma  
Alibaba Group  
maxiao.mx@alibaba-inc.com

Kai Sun  
Alibaba Group  
luchen.sk@alibaba-inc.com

Hang Xiang  
Alibaba Group  
xingzhi.xh@alibaba-inc.com

Xiaoqiang Zhu  
Alibaba Group  
xiaoqiang.zxq@alibaba-inc.com

Hongsong Li  
Alibaba Group  
hongsong.lhs@alibaba-inc.com

Xiaozhong Liu  
Indiana University Bloomington  
lizu237@indiana.edu

ABSTRACT

Anomaly detection is facing with emerging challenges in many important industry domains, such as cyber security and online recommendation and advertising. The recent trend in these areas calls for anomaly detection on time-evolving data with high-dimensional categorical features without labeled samples. Also, there is an increasing demand for identifying and monitoring irregular patterns at multiple resolutions. In this work, we propose a unified end-to-end approach to solve these challenges by combining the advantages of Adversarial Autoencoder and Recurrent Neural Network. The model learns data representations cross different scales with attention mechanisms, on which an enhanced two-resolution anomaly detector is developed for both instances and data blocks. Extensive experiments are performed over three types of datasets to demonstrate the efficacy of our method and its superiority over the state-of-art approaches.

KEYWORDS

Anomaly Detection, Adversarial Autoencoder, High-dimensional Data

1 INTRODUCTION

Anomaly detection aims at identifying outliers or irregular patterns which are inconsistent with the majority of data. It can provide a wide range of applications, from capturing rare events or unusual observations to protecting a complex system against failures or attacks. Recent trend in many important industrial domains, such as online recommendation and advertising (as illustrated in Figure 1), online financial service and cyber security, has set four unprecedented challenges for anomaly detection. First, as data is changed with time, there is no gold standard for anomalous data across all time periods. Second, labeled anomalous samples are rarely available. Third, data format can be very complex, for example, a compound of attributes consisting of categorical ids with extremely high dimension. The sparse, sophisticated and noisy couplings among massive features make it very difficult to recognize the underlying patterns through handcrafted rules or feature engineering. Fourth, a systematic monitoring may require detecting anomalous events at different resolutions for different needs. The data patterns can vary with scales. Although straightforward, aggregating small-scale detection results for larger-scale detection is not guaranteed to be effective. For example, detecting collective patterns such as phase distortion.

Deep learning has drawn immense attention in the field of anomaly detection. Deep neural network has the potential to automatically learn complex feature representations, thus making it possible to train an anomaly detector in an end-to-end fashion with less non-trivial expertise feature engineering. Inspired by Generative Adversarial Network (GAN) [13] and Adversarial Autoencoder [20], some researches have been reported to adversarially train a pair of neural networks (generator and discriminator) through unsupervised or semi-supervised learning to construct an anomaly detector [1, 23, 31]. The networks’ losses, especially adversarial loss and reconstruction loss, are used to identify anomalies. By design, normal samples should follow the distribution close to the majority of the training data, and thus obtain lower losses than anomalous samples. Assuming that the training data is normal, this type of approaches do not require labeled anomalies for training. They are able to solve the second challenge (lack of label) listed above but do not cover the other three.
In this paper, following the emerging idea of adversarially learned anomaly detection, we present an adversarial multiscale anomaly detector (AMAD) to tackle the aforementioned challenges in an end-to-end manner. We train a pair of deep encoder-decoder generator and discriminator to fit the normal patterns of the unlabeled training data (Challenge 2), and using a compound loss as anomaly score for inference. We combine sequential and hierarchical representation learning to detect anomalies at two different scales (Challenge 4) for time-evolving high-dimensional categorical data (Challenge 1,3).

The main contributions of this work are highlighted as follows 1:

- To the best of our knowledge, AMAD is the first unified end-to-end approach to tackle the aforementioned important challenges. Especially, our work is the first attempt to extend adversarial anomaly detector to the scenario of complex high-dimensional categorical data.
- We introduce a multiscale data representation learning mechanism. Patterns are extracted and inspected cross a range of scales, from single features of an individual instance up to data blocks. This produces an enhanced two-resolution anomaly detector for both individual instances and data blocks.
- We report extensive experiments on three types of datasets, validating that our model outperforms the state-of-arts notably. Moreover, we conduct ablation studies to prove the efficacy of the key components in our model.

2 RELATED WORKS

Deep learning has been widely applied in all research topics such as ranking [12], graph mining [11] and text generation [30], etc. As a fundamental one, anomaly detection has been extensively studied via unsupervised or semi-supervised deep approaches. iForest [18], one of the most famous approaches, utilizes a tree-based structure to split data randomly and ranks data points as anomalous based on how easy they get isolated. Affiliated with Support Vector Machine (SVM) family, one-class SVM classifiers [5, 27, 28] use designed kernels to project data to a latent space and search for a best hyperplane to set anomalies apart. Derived from these works, kernel-based one-class classification is further combined with deep neural network [4, 22] to automatically extract useful features from massive complex data.

Deep learning attracts increasing attentions for the past decade. As a basic type of deep learning framework, autoencoder has already widely applied on anomaly detection [2, 3, 25, 32]. It learns to compress the input data with multiple hidden layers and reconstruct the input data through an encoding-decoding mechanism. Trained solely on normal data, autoencoder fails to reconstruct anomalous sample and produces large reconstruction error that can be used to identify anomaly. Furthermore, an autoencoder ensemble with adaptive sampling is proposed to improve the robustness on noisy data [6].

Recently, Generative Adversarial Networks rise up as a popular track in deep learning [15, 23, 26]. Typically, a GAN-based model consists of two parts, i.e., generator and discriminator. The generator learns a representation to resemble the original input data, while the discriminator is trained to distinguish between the resembled and original inputs. The adversarial training enhances the model’s ability of learning the distribution of input normal data, and is proven to be very effective for identifying anomalous or novel data.

Combining GAN and autoencoder, Adversarial Autoencoder [20] offers an alternative way for unsupervised or semi-supervised anomaly detection. Unlike GAN approaches which learns a distribution to generate discrete samples, Adversarial Autoencoder uses autoencoder as the generator to learn to resemble data. By mapping the input to latent space and remapping back to input data space (reconstruction), it enables not only better reconstruction but also control over latent space [8, 21]. Taking this track of thoughts, BiGAN [9] and ALI [10] both apply variational autoencoder as the generators in their models to optimize the distribution of normal data. Two following works [1, 31] combine both GAN and Adversarial Autoencoder components to jointly train an anomaly detector and use the reconstruction errors as the criteria to judge whether testing data is anomalous or not.

3 METHOD

3.1 Overview

The framework of our model is sketched in Figure 2. Our approach adversarially trains an anomaly detection model on unlabeled data, with the assumption that the training data is normal (at least mostly normal). The input data for the model is of a hierarchical four-level structure (also illustrated in Figure 2). We use the model to detect anomalies at the top two levels (instance and block). Since the model is trained to fit the distribution of normal data, the anomalies should have higher loss than normal data. Therefore, we use the loss to infer anomalies [1, 15, 23, 26, 31].

In the following sections, we first introduce the multiscale representation learning across different levels. Second, we describe the adversarial learning architecture. In the end, we explain how we train the model and use the model for inference.

3.2 Multiscale Representation Learning

Our model hierarchically learns representations for the structured input data, from feature, attribute, instance, up to instance block level. We implement attention mechanism [17, 29, 33] to summarize informations with distributed weights of importance to form the
next-level representation, so that most important informations are extracted to the high level.

3.2.1 Feature and Attribute Representation. For the input layer, sparse embedding is implemented to embed each categorical feature to a fixed-size dense vector \( \mathbf{v}^F \), which is automatically learned during the training process. For each attribute, its representation vector \( \mathbf{v}^A \) is extracted from all the embedding vectors of its input feature collection \( \{ \mathbf{v}^F_1, \ldots, \mathbf{v}^F_{N^A} \} \) with a self-attention mechanism [17]:

\[
\begin{align*}
\mathbf{e}^F_i &= (\mathbf{u}^F)^T \tanh(\mathbf{W}^A \mathbf{v}^F_i + b^A), \\
\mathbf{a}^F_i &= \frac{\exp(\mathbf{e}^F_i)}{\sum_{j=1}^{N^A} \exp(\mathbf{e}^F_j)}, \\
\mathbf{v}^A &= \sum_{i=1}^{N^A} \mathbf{a}^F_i \mathbf{v}^F_i,
\end{align*}
\]

where \( \mathbf{W}^A, \mathbf{u}^F \) and \( \mathbf{u}^F \) are trainable weight matrix, bias vector and attention vector, respectively. \( N^A \) denotes the number of attributes. \( \mathbf{v}^F_i, \mathbf{e}^F_i \) and \( \mathbf{a}^F_i \) denote the embedding vector, attention score, and normalized attention score of the \( i \)th feature, respectively.

3.2.2 Instance Representation. Based on the attribute vectors, we construct the higher-level representation for each input instance from two channels, i.e., self representation and relative representation against the previous data block.

The self representation vector, \( \mathbf{v}^S \), is extracted from the instance’s attribute representations \( \{ \mathbf{v}^A_1, \ldots, \mathbf{v}^A_{N^A} \} \):

\[
\begin{align*}
\mathbf{e}^A_i &= (\mathbf{u}^A)^T \tanh(\mathbf{W}^A \mathbf{v}^A_i + b^A), \\
\mathbf{a}^A_i &= \frac{\exp(\mathbf{e}^A_i)}{\sum_{j=1}^{N^A} \exp(\mathbf{e}^A_j)}, \\
\mathbf{v}^S &= \sum_{i=1}^{N^A} \mathbf{a}^A_i \mathbf{v}^A_i,
\end{align*}
\]

where \( \mathbf{W}^A, \mathbf{u}^A \) and \( \mathbf{u}^A \) are trainable weight matrix, bias vector and attention vector, respectively. \( N^A \) denotes the number of attributes. \( \mathbf{v}^A_i, \mathbf{e}^A_i \) and \( \mathbf{a}^A_i \) denote the embedding vector, attention score, and normalized attention score of the \( i \)th attribute, respectively.

The relative representation vector, \( \mathbf{v}^R \), is calculated by comparing the instance’s attributes with the previous block vector. It is designed to enable the model to extract instance’s relative patterns against the larger-scale collective patterns of the data:

\[
\begin{align*}
\mathbf{e}^R_i &= (\mathbf{u}^R)^T \tanh(\mathbf{W}^R [f(\mathbf{v}^A_i), \mathbf{v}^{Mem}] + b^R), \\
\mathbf{a}^R_i &= \frac{\exp(\mathbf{e}^R_i)}{\sum_{j=1}^{N^A} \exp(\mathbf{e}^R_j)}, \\
\mathbf{v}^R &= \sum_{i=1}^{N^A} \mathbf{a}^R_i \mathbf{v}^A_i,
\end{align*}
\]

where the square brackets \([ \ldots ]\) denotes the concatenation operation. The transformation function \( f(\cdot) \) is the Leaky ReLU activation function. \( \mathbf{W}^R, \mathbf{u}^R \) and \( \mathbf{u}^R \) are trainable weight matrix, bias vector and attention vector, respectively. \( \mathbf{e}^A_i \) and \( \mathbf{a}^A_i \) are the attention score and normalized attention score of the \( i \)th attribute, respectively. \( \mathbf{v}^{Mem} \) is the memory vector from the previous data block and its calculation will be described in the next section.

For the output of this module, we concatenate the two latent vectors to form the final representation vector of the instance:

\[
\mathbf{v}^I = \text{batch_norm}([\mathbf{v}^S, \mathbf{v}^R]).
\]

3.2.3 Block Representation. Furthermore, we go beyond instance level and implement a Recurrent Neural Network (RNN) cell to capture the long-term collective patterns of the sequential instances. For each data block, the representation can be calculated as:

\[
\mathbf{v}^B_i = \text{RNN}(f(\mathbf{v}^I), \mathbf{v}^B_{i-1}), i = 1, \ldots, N^B,
\]

where \( N^B \) is the instance number in the block.

The last hidden state, denoted by \( \mathbf{v}^B \), contains the latest and most information about the data’s collective patterns over time. For this reason, we use \( \mathbf{v}^B \) as a representation for the block to improve the block-level anomaly detection (will be revisited in the following paragraphs). Moreover, it is also used as both the memory vector \( \mathbf{v}^{Mem} \) (in Eq. 3) and the initial hidden state for the next block.

3.3 Adversarial Learning

Following the idea of Adversarial Autoencoder, we build an adversarial learning architecture to learn the intrinsic patterns of the training
data for both instance and block levels. On one hand, the encoder-decoder generator part learns to generate resembled representations of the inputs. In this way, cycle consistency [31] is enforced in the latent space. On the other hand, the discriminator tries to distinguish the real and resembled representations.

3.3.1 Instance Generator. We use an autoencoder to generate resembled instance vectors. The autoencoder first encodes the instance representation vector \( v^I \) into a hidden space, and subsequently decodes it back to reconstruct a representation vector \( v^{I*} \):

\[
\begin{align*}
\sigma^{enc} &= W^{enc} f(v^I) + \Delta + b^{enc}, \\
v^{I*} &= f(W^{dec} f(\sigma^{enc}) + b^{dec}) - \Delta,
\end{align*}
\]

where \( Ws \) and \( bs \) are the trainable weights and biases, respectively.

The performance of autoencoder is vulnerable to the noise in training data [32]. In order to get a more robust model, we add a standard Multivariate Gaussian random noise \( \Delta \sim N(0, E) \) into the encode-decode process, where \( E \) is the identity matrix and \( d \) is the dimension of instance vector.

3.3.2 Block Generator. To introduce adversarial learning for long-term patterns, we also reconstruct a resembled vector per block for \( v^B \):

\[
v^{B*} = \text{RNN}(f(v^I), v^{B*}_i), i = 1, ..., N^B. \tag{7}
\]

To be consistent with the calculation of real block vector \( v^B \), here, we don’t train the weight and bias for the RNN cell, but directly copy the values of the corresponding parameters used for Eq. 5. Similarly, the last hidden state is taken as the final resembled vector of the current block, denoted by \( v^{B*} \).

3.3.3 Discriminator. Following the standard setting of binary classification, we build two one-layer neural network classifiers for both instance and block levels:

\[
g^I = \sigma(W^I x^I + b^I) \quad \text{with} \quad x^I \in \{v^I, v^{I*}\} \tag{8}
\]

and

\[
g^B = \sigma(W^B x^B + b^B) \quad \text{with} \quad x^B \in \{v^B, v^{B*}\}, \tag{9}
\]

where \( Ws \) and \( bs \) are the trainable weights and biases, respectively. \( \sigma(\cdot) \) denotes sigmoid activation function.

3.4 Training and Inference

3.4.1 Training. In the training stage, we assume all training data are normal data to train our model in the unsupervised manner. As generative adversarial training is hard to converge, we don’t minimize the generator loss and discriminator loss at the same time. Instead, we minimize the two losses in an alternative process: first holding the discriminator loss \( L_D \) and minimizing generator loss \( L_G \) for several steps, and then minimizing discriminator \( L_D \) with the generator loss \( L_G \) being held.

For the generator loss, we use the sigmoid cross entropy [7] between real and resembled vectors

\[
L_G^I = \sigma(v^I)^T \log(\sigma(v^{I*})) + (1 - \sigma(v^I))^T \log(1 - \sigma(v^{I*})) \tag{10}
\]

as the instance generator loss, and

\[
L_G^B = \sigma(v^B)^T \log(\sigma(v^{B*})) + (1 - \sigma(v^B))^T \log(1 - \sigma(v^{B*})) \tag{11}
\]

as the block generator loss. The total generator loss to minimize is the sum of block generator loss and the average of its \( N^B \) instance generator losses:

\[
L_G = \frac{1}{N^B} \sum_{i=1}^{N^B} L_G^I, + L_G^B. \tag{12}
\]

For the discriminator loss, under standard setting of binary classification, we also use cross entropy based on the output of Equations 8 and 9

\[
L_D^I = y^I \log(y^{I*}) + (1 - y^I) \log(1 - y^{I*}) \tag{13}
\]

as the instance discriminator loss, and

\[
L_D^B = y^B \log(y^{B*}) + (1 - y^B) \log(1 - y^{B*}) \tag{14}
\]

as the block discriminator loss. \( y \in \{y^I, y^B\} \) is defined as: \( y = 1 \) for real vectors and \( y = 0 \) for resemble vectors. In each optimization step, the total discriminator loss of a data block to minimize is:

\[
L_D = \frac{1}{N^B} \sum_{i=1}^{N^B} L_D^I, + L_D^B. \tag{15}
\]

3.4.2 Inference. For inference, we use compound loss as the output anomaly score to measure the degree of abnormality. Because the model lowers the total loss by learning to fit normal data patterns during training. Abnormal data will produce higher loss since the model fails to fit abnormal patterns. The anomaly score of an instance is given by:

\[
z^I = L_G^I + \beta \cdot L_D^I. \tag{16}
\]

The anomaly score for a block is calculated by including both the block-level losses and the average instance anomaly score within the block:

\[
z^B = L_G^B + \beta \cdot L_D^B + y \cdot \frac{1}{N^B} \sum_{i=1}^{N^B} z^I_i. \tag{17}
\]

Two weight parameters \( \beta \) and \( y \) are introduced to balance the influences from the different terms.

4 EXPERIMENT SETUP

4.1 Datasets

Three datasets are utilized to illustrate the performance of the proposed method. Their statistics are shown in Table 1, and more details are described in Appendix.

- **Synthetic dataset**: The Synthetic data is generated by adding random noises to multi-dimensional zigzag signals of discrete integers. The anomalies are constructed by either randomly generating numbers or randomly copying training instances.

- **Public dataset**\(^3\): It is a public dataset about positions in the ‘connect-4’ game. We use the instances labelled with ‘win’ as normal data and the instances labelled with ‘loss’ as anomaly data. Note that there is no sequential relation within the data.

- **Industrial dataset**\(^4\): The Industrial dataset is constructed by user behavior data from a real-world online recommendation system, which is very important for many tasks, such as click-through rate prediction [19, 33]. The instances are collected over 10 consecutive

\(^3\)https://archive.ics.uci.edu/ml/datasets/Connect-4

\(^4\)The Industrial dataset is published at https://tianchi.aliyun.com/dataset/dataDetail?dataId=27665.
days, and stored in the order of timestamp. As illustrated in Figure 1, each instance consists of multiple attributes about user’s past behaviors, e.g., clicked items in the past 3 days, favorite brands in the past week, etc. Each attribute contains a group of categorical ids in representation of the corresponding items, brands, etc. It’s impractical to get a dataset with enough well labeled anomalies from real-world production. Thus, we mimic anomalies by simulating real conditions (attributes are polluted by errors in upstream data pipeline). The anomalies are generated by deleting the records of a random selected attribute, or replacing the records with random ids.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Dimension</th>
<th>#Attribute</th>
<th>#Normal</th>
<th>#Anomaly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthetic</td>
<td>30</td>
<td>3</td>
<td>10,000</td>
<td>1,000</td>
</tr>
<tr>
<td>Public</td>
<td>192</td>
<td>64</td>
<td>44,000</td>
<td>4,000</td>
</tr>
<tr>
<td>Industrial</td>
<td>440,512</td>
<td>8</td>
<td>783,000</td>
<td>25,000</td>
</tr>
</tbody>
</table>

Table 1: Statistics of three datasets used in the paper. The dimension denotes the total number of all the distinct categorical feature IDs for the entire dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Reference</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCSVM</td>
<td>[27]</td>
<td>SVM</td>
</tr>
<tr>
<td>iForest</td>
<td>[18]</td>
<td>Tree ensemble</td>
</tr>
<tr>
<td>RDA</td>
<td>[32]</td>
<td>Autoencoder</td>
</tr>
<tr>
<td>ALAD</td>
<td>[31]</td>
<td>GAN</td>
</tr>
<tr>
<td>GANomaly</td>
<td>[1]</td>
<td>GAN</td>
</tr>
<tr>
<td>ALOCC</td>
<td>[23]</td>
<td>GAN</td>
</tr>
</tbody>
</table>

Table 2: Baseline methods in this paper.

For the Public dataset, the normal data are randomly split for either training or testing, while the testing normal data is mixed with the anomalous data to form the final testing set. For the Synthetic and Industrial datasets, the former part (the majority) of the normal data is used for training, whereas the last small portion is used separately and mixed with anomaly samples for testing. To better cover the high-dimension space outside the normal data and test the models’ performance more efficiently, we include a large ratio of anomalies in the training data. We add a random noise into the autoencoder approach.

5 RESULTS

In this section, we report the experimental results to demonstrate our approach’s superiority over the other methods. Moreover, we present studies to verify the effects of the important characteristics of our model.

5.1 Performance of the Full Model

We run our model ten times and report the average evaluation results on instance-level anomaly detection and block-level anomaly detection in Tables 3 and 4. To verify our model’s superiority, we calculate the performance differences between our model and the best baseline on each metric for all the runs, and apply a T-test to check whether the performance difference is significantly above 0 or not.

We can find that the GAN-based models perform best among the baselines, while our model outperforms all the baselines with respect to all the metrics and datasets. Moreover, our model displays larger advantages for block-level detection. Clearly, the block-level anomaly detection gets more benefits from our unified multiscale approach.

5.2 Random Noise in Autoencoder

The performance of autoencoder can be deteriorated due to the noises in the training data. We add a random noise into the autoencoder (Eq. 6), to make it more robust. To check the utility of the added noises, we retrain an ablated model by removing the $\Delta$ in Eq. 6. As shown by the results in the first rows (–Noise) of Tables 5 and 6, adding noise clearly improves the generalization performance for all the tests.
5.3 Relative Representation of Instance

We introduce the relative representation $v^R$ (Eqs. 3 and 4) to improve the pattern recognition. To justify this, we retrain an ablated model (–RelRep) without this module, i.e., deleting Eq. 3 and removing $v^R$ from Eq. 4. It leads a big drop in the performance, as shown in the second rows of Tables 5 and 6. Through comparing the instance with block representation, $v^R$ notably enriches the information extracted at high level.

5.4 Block Loss for Block-Level Detection

To testify the effect of the block loss for detecting anomalous blocks. In consistent with baseline approaches, we remove the first two terms in Eq. 17, and only use the average instance anomaly score. As listed in the third row of Table 6 (–BlockLoss), the performance drops drastically down to the level very close to the baselines! The block loss adds detection of the collective patterns to our model, which is of great importance for detecting block-level anomaly.
span of resolutions (from instance and block level to hour and day levels), large-scale distributed data process.

REFERENCES


7 APPENDIX

7.1 Details of Datasets

**Synthetic dataset:** We initialize the first instance with three categorical ids ‘0,10,20’, and then generate the following instances by add 1 on each ID of the previous instance. If the ID surpasses 30, it will be subtracted by 30 and substituted by the remainder. Noises are introduced to the deterministic signals, by randomly selecting 10% IDs and adding random noises $\in \{-1, 1\}$ onto their original values. This process is repeated 220 times with a period of 50, generating 11000 normal instances. We use the first 9000 normal instances as the training data and the remaining 2000 for testing. In the testing set, we randomly select 1000 instance and replace them with anomalies. The anomalies are constructed by either randomly generating IDs or copying randomly selected training instances.

**Public dataset:** It is about positions in the ‘connect-4’ game. Each feature in an instance refers to one of three choices (taken, not taken, blank) on a position. Each instance refers to a possible choice permutation. We use the instances labelled with ‘win’ as normal data and the instances labelled with ‘loss’ as anomaly data. We randomly select 40000 normal instance for training, while randomly mix the other 4000 with the 4000 anomalies to form the testing set.

**Industrial dataset:** The Industrial dataset is constructed by user behavior data from our online recommendation system. The user behavior record is updated according to user’s latest behaviors. Whenever the system receives an impression request from a user, a instance is generated. The data is collected over 10 consecutive days, and stored in the order of timestamp. All the real-world data is assumed to be normal. We use the first 758000 normal instances as the training data and the remaining 50000 for testing. In the testing set, we randomly select 25000 instances and replace them with anomalies. The anomalies are generated by deleting the records under a random selected attribute, replacing the records with random IDs.