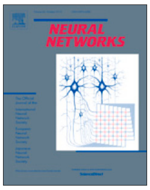
Neural Networks 165 (2023) 868–883

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet



TCGAN: Convolutional Generative Adversarial Network for time series classification and clustering

Fanling Huang [*], Yangdong Deng

School of Software, Tsinghua University, Beijing, China

ARTICLE INFO

Article history:
Received 25 July 2022
Received in revised form 4 June 2023
Accepted 25 June 2023
Available online 30 June 2023

Keywords:
Time series
Classification
Clustering
Generative Adversarial Networks
Deep Neural Networks
Representation learning

ABSTRACT

Recent works have demonstrated the superiority of supervised Convolutional Neural Networks (CNNs) in learning hierarchical representations from time series data for successful classification. These methods require sufficiently large labeled data for stable learning, however acquiring high-quality labeled time series data can be costly and potentially infeasible. Generative Adversarial Networks (GANs) have achieved great success in enhancing unsupervised and semi-supervised learning. Nonetheless, to our best knowledge, it remains unclear how effectively GANs can serve as a general-purpose solution to learn representations for time series recognition, i.e., classification and clustering. The above considerations inspire us to introduce a Time-series Convolutional GAN (TCGAN). TCGAN learns by playing an adversarial game between two one-dimensional CNNs (i.e., a generator and a discriminator) in the absence of label information. Parts of the trained TCGAN are then reused to construct a representation encoder to empower linear recognition methods. We conducted comprehensive experiments on synthetic and real-world datasets. The results demonstrate that TCGAN is faster and more accurate than existing time-series GANs. The learned representations enable simple classification and clustering methods to achieve superior and stable performance. Furthermore, TCGAN retains high efficacy in scenarios with few-labeled and imbalanced-labeled data. Our work provides a promising path to effectively utilize abundant unlabeled time series data.

1. Introduction

Time series have become increasingly valuable in various domains, such as biology, meteorology, finance, medicine, and the Internet of Things (IoT). Accurately recognizing different types of time series is an important problem, but remains challenging because the time series data tend to be noisy, dynamic, and highly problem-specific (Bagnall, Lines, Bostrom, Large, & Keogh, 2017; Fawaz, Forestier, Weber, Idoumghar, & Muller, 2019; Javed, Lee, & Rizzo, 2020; Middlehurst et al., 2021).

In recent years, Deep Neural Networks (DNNs), in particular Convolutional Neural Networks (CNNs), have been demonstrated to be effective in time series classification due to their capability to learn hierarchical representations (Fawaz et al., 2019; Ismail Fawaz et al., 2020; Tang et al., 2022). However, most existing methods view representation learning as an intrinsic part of a supervised DNN, which can tend to be unstable when labeled data is limited. To illustrate this issue, we trained the Fully Convolutional Network (FCN) (Fawaz et al., 2019) on 85 UCR datasets (Chen et al., 2015) for 5 repetitions with different random seeds. We found 18 datasets with standard deviations greater than 0.1. Fig. 1 shows the five datasets with standard deviations greater than 0.2. In practice the problem is difficult to resolve by collecting more training data because gathering high-quality labeled time series data can be costly or potentially infeasible. There are two main reasons for this. Firstly, human beings are less sensitive to time series, and thus the labeling process can be tedious and error-prone. Secondly, the novelties of interest (e.g., system crashes and physical lesions) in time series are innately low-probability events.

Generative models learn discriminative representations in an unsupervised manner, showing promise to alleviate the shortage of labeled data (Längkvist, Karlsson, & Loutfi, 2014). In particular, Generative Adversarial Nets (GANs) have achieved great success in boosting unsupervised and semi-supervised learning (Creswell et al., 2018; Goodfellow et al., 2014). GANs have already found impressive applications in computer vision (Creswell et al., 2018; Isola, Zhu, Zhou, & Efros, 2017; Jenni & Favaro, 2018). Few studies have applied GANs for general time series generation (e.g., TimeGAN Yoon, Jarrett, & Van der Schaar, 2019 and CotGAN Xu, Wenliang, Munn, & Acciaio, 2020) and specific time series mining problems (e.g., anomaly detection Li, Chen, Goh, & Ng, 2018 and imputation Luo, Zhang, Cai, & Yuan, 2019).

[*] Corresponding author.
E-mail addresses: huangfanling.2015@tsinghua.org.cn (F. Huang), dengyd@mail.tsinghua.edu.cn (Y. Deng).

https://doi.org/10.1016/j.neunet.2023.06.033
0893-6080/

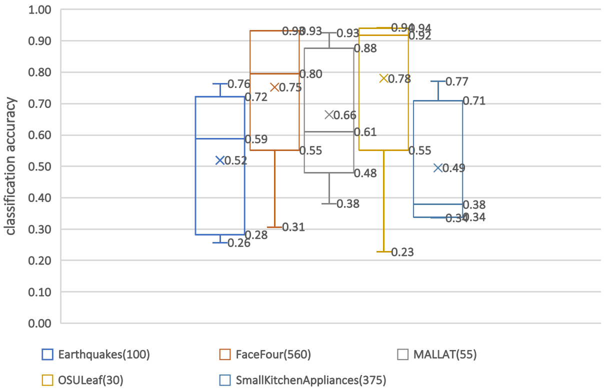


Fig. 1. FCN suffers from high variance when labeled data are limited. FCN was trained on each dataset 5 times with different random seeds. The number in bracket next to the data name is the size of the training set.

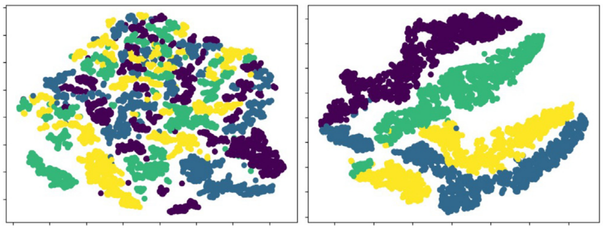


Fig. 2. t-SNE plots for (left) the raw time series and (right) the TCGAN representations of the Two_Patterns dataset. Each point corresponds to a time series. Colors annotate the four classes.

However, these advanced GANs depend primarily on autoregressive models like Recurrent Neural Networks (RNNs), which learn in a sequential manner and are theoretically less efficient than CNNs that can be easily parallelized over time (Bai, Kolter, & Koltun, 2018). Applying convolutional GANs for efficient sequence modeling has been explored in some task-specific studies, such as irregular sampling time series imputation (Ramponi, Protopapas, Brambilla, & Janssen, 2018), speech generation (Ye, Hu, & Xu, 2020), human motion prediction (Cui, Sun, Kong, Zhang, & Li, 2021) and time series sampling (Dahl & Sørensen, 2021). To the best of our knowledge, it remains unclear how effectively GANs can serve as a general-purpose solution to learn representations from diverse unlabeled time series data and then support time series mining tasks like classification and clustering.

Fusing the two sides above, we introduce a Convolutional GAN to learn representations from unlabeled time series data, dubbed as Time-series Convolutional GANs (TCGANs). These learned representations can then be used by simple recognition models to achieve stable and superior performance. Specifically, TCGAN consists of two one-dimensional CNNs (i.e., a generator and a discriminator). Each CNN learns through playing an adversarial game with the other. The generator attempts to generate statistically realistic time series, while the discriminator endeavors to distinguish real time series from the generated fake ones. Parts of the trained discriminator are then reused to form an independent representation encoder. As illustrated in Fig. 2, the encoder is capable of transforming raw time series into an alternative data space where instances of different classes can be more easily differentiated.

The major contributions of this paper can be summarized as follows:

- We propose TCGAN, a GAN-based unsupervised time series representation learning framework, which can be seamlessly used with time series classification and clustering.
- We conduct comprehensive experiments on synthetic datasets and 85 time series datasets from the UCR archive. The experimental results show that: (1) TCGAN significantly outperforms leading time-series GANs in terms of generation performance, classification accuracy, and runtime. (2) TCGAN representations can enable simple classifiers to obtain better accuracy than unsupervised and supervised CNNs. TCGAN classifiers can also obtain a decent level of performance even where labeled data are highly limited and imbalanced. (3) TCGAN representation preserves the pairwise similarities between time series, which allows the distance-based clustering method to achieve higher accuracy compared with most leading competitors
- To motivate more attention on applying generative models, particularly GANs, to serve the time series mining community, we open the source code at https://bitbucket.org/Lynn1/tcgan.

2. Related work

2.1. Time series classification

Time series classification is one of the most pervasive research problems in the data mining domain. Existing approaches can be divided into three categories: distance-based, feature-based, and ensemble approaches. The distance-based method integrates a distance or similarity measure into the nearest neighbor classifier. The 1-nearest neighbor (1NN) with Dynamic Time Warping (DTW) elastic distance established a hard-to-beat baseline (Lines & Bagnall, 2015). The feature-based method focuses on distilling salient representations from raw data for accurate classification. Conventional works have mined various time series features, for example, measures of time series characteristics (skewness, trend, seasonality, etc.) Christ, Braun, Neuffer, and Kempa-Liehr (2018) and salient subsequences like shapelets (Hills, Lines, Baranauskas, Mapp, & Bagnall, 2014). Recent DNN classifiers resort to their capability to automatically learn hierarchical representations (Fawaz et al., 2019). There is a growing consensus that no single setup can produce universally superior performance on diverse time series datasets, prompting ensemble methods, like InceptionTime (Ismail Fawaz et al., 2020), HiveCOTE (Middlehurst et al., 2021), Omni-Scale CNNs (Tang et al., 2022) and MultiRocket (Tan, Dempster, Bergmeir, & Webb, 2022), to pursue the State-Of-The-Art (SOTA) by exhausting and fusing diverse time series representations.

Convolutional Neural Networks (CNNs) have been demonstrated to be effective and efficient in modeling sequential data (Bai et al., 2018; Fawaz et al., 2019; Tang et al., 2022). However, single-supervised DNN tends to be unstable in the absence of labeled data. An ensemble strategy is often used to obtain a stable performance. For example, InceptionTime (Ismail Fawaz et al., 2020) unifies five Inception networks (i.e., CNNs) to reduce the variance of evaluation results. To achieve the SOTA accuracy, Tang et al. (2022) fuses five OS-CNNs and equips each OS-CNN with the proposed Omni-Scale block (OS-block) that integrates multiple receptive fields with different kernel sizes. Furthermore, some works have investigated data augmentation (Wang, Li, Wang, & Zheng, 2021), transfer learning (Kashiparekh, Narwariya, Malhotra, Vig, & Shroff, 2019), meta learning (Narwariya, Malhotra, Vig, Shroff, & Vishnu, 2020), and self-supervised learning (Eldele et al., 2022) to make DNNs applicable in scenarios where labeled time series data is lacking.

Differently, we focus on generative models, which can learn discriminative representations in an unsupervised manner (Längkvist et al., 2014) but have received less attention in the time series classification community. Specifically, we leverage GANs to learn time series representations from abundant unlabeled data. The resulting representations can enable simple classifiers to obtain more superior and stable accuracy compared with single-supervised CNNs, even where labeled data are highly limited and imbalanced.

2.2. Time series clustering

Clustering aims to group unlabeled data by uncovering complex distributions and structures inherent in the data. Existing time series clustering methods could be categorized into raw-data-based and feature-based methods. The raw-data-based approach focuses on devising an appropriate distance or similarity measure that captures the shape features in time series. Typically, the k-shape algorithm proposed in Paparrizos and Gravano (2017) is a superior raw-data-based algorithm that uses a normalized version of the cross-correlation as distance measure. The feature-based approach first represents a raw time series with a feature vector and then applies a clustering algorithm on the extracted feature vectors. For example, Lei, Yi, Vaculin, Wu, and Dhillon (2019) proposes the Similarity PreservIng RepresentAtion Learning (SPIRAL) algorithm and feeds the resulting representations into a k-means algorithm with DTW or Move-Split-Merge (MSM) as a distance measure. Interested readers could refer to these time series clustering surveys (Aghabozorgi, Shirkhorshidi, & Wah, 2015; Javed et al., 2020) for more details. Our method is feature-based. Our method derives a transformation space where the patterns of different groups of data become apparent, and the pairwise similarities between time series are well preserved. As a result, simple clustering methods like k-means with Euclidean distance can achieve superior performance.

2.3. Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs), firstly proposed by Goodfellow et al. (2014), are SOTA generative models amenable for unsupervised and semi-supervised learning. In the image domain, there has been a large body of research on developing new architectures (Radford, Metz, & Chintala, 2015), optimization techniques (Gulrajani, Ahmed, Arjovsky, Dumoulin, & Courville, 2017), and applications (Isola et al., 2017) under the framework of GANs. We refer the reader to recent surveys (Brophy, Wang, She, & Ward, 2021; Creswell et al., 2018) for more information.

The development of GANs for use on time series data is still in an early stage (Brophy et al., 2021). Existing works mainly focus on generating realistic time series. TimeGAN (Yoon et al., 2019) proposes to generate and discriminate within a jointly optimized embedding space, as well as combine unsupervised adversarial training with a supervised teacher-forcing component to capture the autoregressive natures of time series. Causal Optimal Transport GAN (COT-GAN) (Xu et al., 2020) introduces a new adversarial objective based on Causal Optimal Transport (COT) theory to model both complex spatial structures and temporal dependences in time series. Time-series Generation by Contrastive Imitation (TimeGCI) (Jarrett, Bica, & van der Schaar, 2021) captures both the conditional dynamics of (stepwise) transitions and the joint distribution of (multi-step) trajectories in time series. Pei et al. (2021) pays special attention to generating long sequences with variable lengths and missing values. There are also some domain-specific or task-specific attempts, for example, Wiese, Knobloch, Korn, and Kretschmer (2019) proposes Quant GAN for financial time series generation, Li et al. (2018) works on time series anomaly detection, and Luo et al. (2019) targets multivariate time series imputation.

The above time-series GANs mainly depend on autoregressive models like RNNs, which are less efficient than parallelizable CNNs (Bai et al., 2018). Some studies have applied convolutional layers for efficient sequence modeling, but they only focused on limited applications and did not open the source code for further exploration. Ramponi et al. (2018) uses convolutional layers in a conditional GAN for irregular sampling time series imputation and conducts experiments on 1 synthetic and 3 UCR datasets. Ye et al. (2020) proposes a temporal dilated convolutional GAN for speech generation and evaluates the model on one simulated speech dataset. Cui et al. (2021) focuses on human motion prediction and uses temporal convolutions in the GAN structure for efficient sequence-to-sequence modeling. Dahl and Sørensen (2021) uses a GAN embedded with temporal convolutions in a bootstrap-like method for time series, and conducts experiments on a simulated AR(1) time series process and a U.S. equity dataset. In general, it remains unclear how effective Convolutional GANs are at modeling diverse time series, particularly for classification and clustering tasks

3. Proposed framework

As illustrated in Fig. 3, we propose a Time-series Convolutional Generative Adversarial Network (TCGAN) based classification and clustering framework. Given a training dataset containing C classes. The dataset consists of a labeled subset $\mathbf{X}^L = \{\mathbf{x}^i\}_{i=1}^L$ with labels $\mathbf{Y}^L = \{y^i\}_{i=1}^L$ ($y^i \in \{1, 2, \dots, C\}$) and an unlabeled subset $\mathbf{X}^U = \{\mathbf{x}^i\}_{i=1}^U$. Without loss of generality, we assume the first L samples within $\mathbf{X} = \{\mathbf{X}^L, \mathbf{X}^U\}$ are labeled by $\mathbf{Y}^L$. Each sample $\mathbf{x}^i \in \mathbb{R}^{n*d}$ is a time series of length n and number of variables d. For readability, the superscript i to denote a sample is omitted without ambiguity. As a result, $\mathbf{x}^i = \mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_t, \dots, \mathbf{x}_n]$, $\mathbf{x}_t \in \mathbb{R}^d$ is a vector of observations at timestamp t. First, TCGAN learns from entire training dataset $\mathbf{X} = \{\mathbf{X}^L, \mathbf{X}^U\}$ in an unsupervised manner. Specifically, TCGAN takes the form of a standard GAN consisting of a generator and a discriminator. The generator, $G(\mathbf{z}; \boldsymbol{\theta}_\mathbf{g}) : \mathbf{z} \in \mathbb{R}^{n_z} \rightarrow \hat{\mathbf{x}} \in \mathbb{R}^{n*d}$, a CNN-based sampler with parameters $\boldsymbol{\theta}_\mathbf{g}$, learns to approximate the distribution of real data P_{data}. G takes a random variable $\mathbf{z}$ as input that obeys a predefined prior P_z and generates a fake time series $\hat{\mathbf{x}}$. The discriminator, $D(\widetilde{\mathbf{x}}; \boldsymbol{\theta}_\mathbf{d}) : \widetilde{\mathbf{x}} \in \mathbb{R}^{n*d} \rightarrow p_{real} \in [0, 1]$, a binary classification CNN with parameters $\boldsymbol{\theta}_\mathbf{d}$, outputs a single scalar p_{real} representing the probability that $\widetilde{\mathbf{x}}$ comes from P_{data} rather than $G(\mathbf{z}; \boldsymbol{\theta}_\mathbf{g})$. The generator ($G$) and discriminator ($D$) of TCGAN learn from each other by competing in an adversarial game in absence of any labeled information. Then, an encoder is constructed by reusing parts of the pretrained discriminator. The resulting TCGAN encoder, $E(\mathbf{x}; \boldsymbol{\theta}_\mathbf{e}) : \mathbf{x} \in \mathbb{R}^{n*d} \rightarrow \mathbf{v} \in \mathbb{R}^{n_v}$, transforms each time series $\mathbf{x}$ into a feature vector $\mathbf{v}$ of length n_v for an off-the-shelf linear classification or clustering model. In the following subsections, we will present how to build the proposed framework compatible with time series data.

3.1. TCGAN

We adapt our generator and discriminator architectures from an image synthesis GAN, which was developed through extensive model exploration (Radford et al., 2015). Our networks are based on the one-dimensional CNN for modeling time series data. The filters exhibit only one dimension (time) instead of two dimensions (width and height) in images. In comparison with the recent time-series GANs (Xu et al., 2020; Yoon et al., 2019)

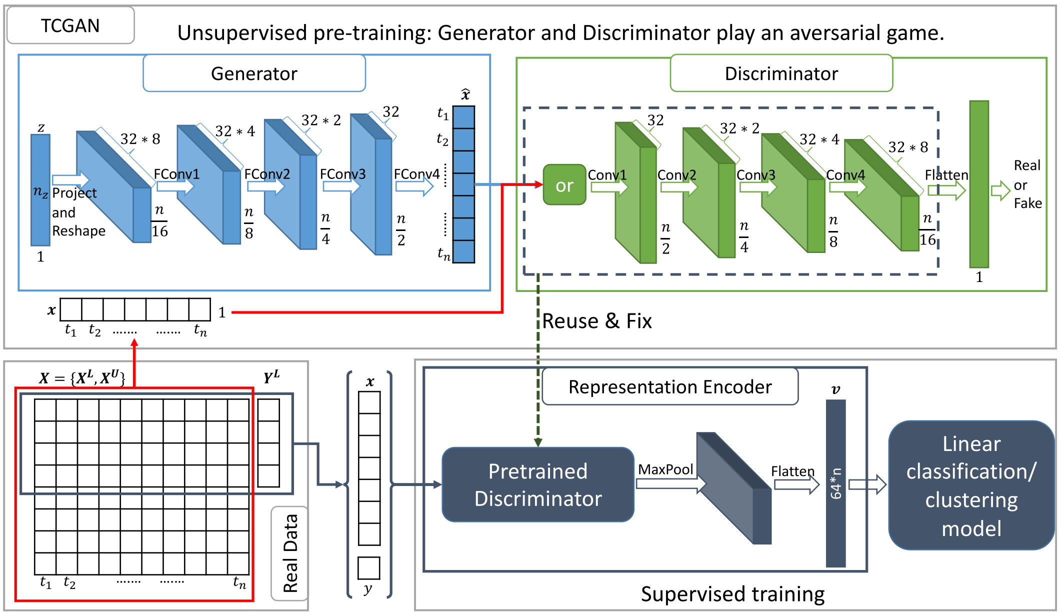


Fig. 3. TCGAN-based classification and clustering framework.

that use an autoregressive model (e.g., RNN) to capture the conditional dynamics of stepwise transitions, $p(\mathbf{x}_t|\mathbf{x}_1, \ldots, \mathbf{x}_{t-1})$, $t \in \{1, 2, \ldots, n\}$, we found modeling entire time series, $p(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, with stacking convolution layers is more efficient and effective. In general, TCGAN has the following advantages: (1) The model can see contextual information from both directions, $\mathbf{x}_t$ is attended to the information from $\mathbf{x}_{[:t-1]}$ and $\mathbf{x}_{[t+1:]}$. (2) Stacking convolution layers can model long dependencies and hierarchical features in time series (Ismail Fawaz et al., 2020). (3) Without modeling time series step by step, CNNs are easy to parallel (Bai et al., 2018).

3.1.1. Generator

The generator, $G(\mathbf{z}; \boldsymbol{\theta}_\mathbf{g}) : \mathbf{z} \in \mathbb{R}^{n_z} \rightarrow \hat{\mathbf{x}} \in \mathbb{R}^{n*d}$, maps a noise sequence $\mathbf{z} \sim P_z$ to a time series $\hat{\mathbf{x}}$ mimicking the real $\mathbf{x} \sim P_{data}$. G has five layers and processes one batch of samples at a time. First, a dense layer with a ReLu activation function projects and reshapes the input to align with the following layers. Next, four one-dimensional Fractionally-strided Convolutional (FConv) layers (Springenberg, Dosovitskiy, Brox, & Riedmiller, 2014) are applied sequentially to generate a batch of time series as output. All FConv layers except the last one have a FConv-BatchNorm-Relu structure (the last layer just performs a FConv operation). The FConv operation uses a strided convolution to replace the deterministic spatial pooling function (e.g., maxpooling) and allows the generator to learn its own spatial up-sampling. The BatchNorm operation (Ioffe & Szegedy, 2015) is deployed to stabilize the learning process and prevent the problem of "mode collapse" in GANs. We do not use any activation function for the last layer because it tends to distort time series. In fact, we do not perform any pre- and post-processing on real or generated data except for applying z-normalization to the primitive time series. All convolutional filters share a common setting, i.e., each filter has a kernel size of 10 and a stride of 2. We found a large kernel size is generally helpful for time series modeling, yet reduces efficiency to some degree. Thus, we choose a moderate value of 10. The stride of 2 results in a 2-times up-sampling on the input after each FConv operation.

3.1.2. Discriminator

The discriminator, $D(\widetilde{\mathbf{x}}; \boldsymbol{\theta}_\mathbf{d}) : \widetilde{\mathbf{x}} \in \mathbb{R}^{n*d} \rightarrow p_{real} \in [0, 1]$, is a FCN for binary classification. Specifically, D consists of six layers to classify each inputted time series as real or fake. The input is a batch of real or fake $(\mathbf{x}/\hat{\mathbf{x}})$ time series sequentially flowing through four one-dimensional Convolutional (Conv) layers. All Conv layers except the first one have a Convolution-BatchNorm-LeakyReLU structure (no BatchNorm operation in the first layer). All convolutional filters have the same setting, i.e., each filter has a kernel size of 10, and a stride of 2. The stride of 2 results in a 2-times down-sampling of the input after each Conv operation. The output of the above process is flattened into a batch of one-dimensional feature vectors for a sigmoid layer to conduct binary classification.

3.1.3. Training

Following Goodfellow et al. (2014), G and D are trained by playing a two-player minimax game with a value function defined as Eq. (1). D attempts to determine whether a sample is from G or the real dataset, while G tries to produce fake data (but mimicking the distribution of the real dataset) to fool D.

$$\begin{aligned}\min_G \max_D V(G, D) = \mathbb{E}_{\mathbf{x}\sim P_{data}}[log(D(\mathbf{x}))]+ \\ \mathbb{E}_{\mathbf{z}\sim P_z}[1 - log(D(G(\mathbf{z})))]\end{aligned} \tag{1}$$

Please refer to Alg. 1 for the algorithmic details. TCGAN is trained with unlabeled data by using the Adam algorithm (Kingma & Ba, 2014) with empirically chosen hyper-parameters. We follow the non-saturating training criterion for G, i.e., training G to maximize $log(D(G(\mathbf{z})))$ instead of minimizing $log[1-D(G(\mathbf{z}))]$ so as to derive stronger gradients for G in early learning (Goodfellow et al., 2014). A straightforward training procedure for GAN is to update G and D once per batch (Goodfellow et al., 2014; Xu et al., 2020). In our practical experience with time series data, we observed that D usually learns much faster than G. A possible reason is that generating time series is more difficult than distinguishing real time series from synthetic time series.

Therefore, it is hard for G to extract signals for improvement from D because all generated samples would be rejected by D with high confidence. In addition, we found that fixing the number of iterations of D and G cannot be applied effectively to a wide variety of time series datasets. Motivated by the above considerations, we adopt an adaptive training strategy to maintain a balance between G and D. Specifically, for each batch, D is updated only if its accuracy in the last batch is no larger than a predefined threshold $\delta \in (0.5, 1.0)$. We found that $\delta = 0.75$ can stabilize the training and derive more reasonable results. It should be noted that the threshold δ is not very sensitive, so another similar value is also permissible.

Algorithm 1 Training TCGAN. Default hyper-parameters: $m = 16, n_{epoch} = 300, P_z = Uniform(-1, 1), n_z = 100, \delta = 0.75, \alpha = 0.0002, \beta_1 = 0.5, \beta_2 = 0.9$.

Input: An unlabeled time series dataset of size n_{sample} and its distribution is represented as P_{data}.
Hyper-parameters: The batch size, m. The number of passes on data, n_{epoch}. The noise prior, P_z, and the length of a noise vector, n_z. The accuracy threshold triggering D to update, δ. Adam hyper-parameters include the learning rate α, the exponential decay rate for the 1st moment estimates β_1 and the 2nd moment estimates β_2.

```
$n_{batch} = \lfloor n_{sample}/m \rfloor$
$acc_{last} = \delta$
**for** $e = 0; e < n_{epoch}; e++$ **do**
    **for** $j = 0; j < n_{batch}; j++$ **do**
        sample $\{\mathbf{x}^{(i)}\}_{i=1}^{m}$ from $P_{data}$
        sample $\{\mathbf{z}^{(i)}\}_{i=1}^{m}$ from $P_z$
        **if** $acc_{last} <= \delta$ **then**
            $Loss_d = -\frac{1}{m}\sum_{i=1}^{m}[log(D(\mathbf{x}^{(i)})) + log(1 - D(G(\mathbf{z}^{(i)})))]$
            $\theta_{\mathbf{d}} = Adam(\nabla_{\theta_{\mathbf{d}}} Loss_d, \theta_{\mathbf{d}}, \alpha, \beta_1, \beta_2)$
        **end if**
        $Loss_g = -\frac{1}{m}\sum_{i=1}^{m} log(D(G(\mathbf{z}^{(i)})))$
        $\theta_{\mathbf{g}} = Adam(\nabla_{\theta_{\mathbf{g}}} Loss_g, \theta_{\mathbf{g}}, \alpha, \beta_1, \beta_2)$
        $acc_{last} = Acc(\{< D(\mathbf{x}^{(i)}), 1 >\}_{i=1}^{m}, \{< D(G(\mathbf{z}^{(i)})), 0 >\}_{i=1}^{m})$
    **end for**
**end for**
```

3.2. Representation encoder

We reuse parts of the discriminator to construct a representation encoder, $E(\mathbf{x}; \boldsymbol{\theta}_{\mathbf{e}}) : \mathbf{x} \in \mathbb{R}^{n*d} \rightarrow \mathbf{v} \in \mathbb{R}^{n_v}$, that transforms each time series into a representation vector. Specifically, we follow common practices in the DNN community to frame the encoder. First, the encoder feeds the input of real time series to the pretrained discriminator and uses the feature maps from the last Conv layer (i.e., Conv4). Then, a max pooling layer with a pooling size of 2 and a stride of 1 is applied to these feature maps. The final representation vector is derived with a flatten layer. We will describe the design details below.

Although the discriminator does not see the real labels, we believe it has learned how to distill salient features from raw data by competing against the generator during the adversarial game. In fact, the discriminator is a supervised FCN that has been demonstrated to be powerful in feature learning (Fawaz et al., 2019; LeCun, Bengio, & Hinton, 2015). Furthermore, the idea that the discriminator learns from a surrogate binary classification task in a GAN architecture is similar to the approach of using surrogate labels to realize self-supervised learning (Doersch & Zisserman, 2017; Jenni & Favaro, 2018). In the computer vision domain Creswell et al. (2018), the GAN discriminator has been demonstrated to be helpful for a variety of downstream tasks. As shown in Fig. 2, time series of different classes do become easier to distinguish in the TCGAN transformation space.

For compact representation, we only reuse outputs from one Conv layer, instead of fusing all layers. We opt to use the last Conv layer because it is widely recognized that the deeper a layer is located in a neural network, the more abstract the underlying representation will be (LeCun et al., 2015).

The superiority of CNNs in time series feature learning has been illustrated in recent literature (Fawaz et al., 2019). In fact, convolution is a well-established method for handling sequential signals (Mallat, 1999). Suppose $\mathbf{x} \in \mathbb{R}^{n*d}$ is a time series of length n and $\mathbf{f} \in \mathbb{R}^{w*d}$ is a filter of length w. Let $(\mathbf{x} * \mathbf{f})$ denote the result of one-dimensional discrete convolution. A general form of performing convolution at a time stamp t is given by Eq. (2) where b_t is an independent bias. The filter is shared by all time stamps to extract time-invariant features across the whole time series. The result of one filter can be viewed as a transformation on the input time series. And thus, one filter detects one type of features. Multiple filters in a Conv layer extract different types of features.

$$(\mathbf{x} * \mathbf{f})[t] = \sum_{i=0}^{w-1} \mathbf{x}_{t+i} \cdot \mathbf{f}_{w-i} + b_t, t \in \{1, 2, \ldots, n\} \tag{2}$$

We use a max-pooling layer right after the Conv feature maps to enable the transformations to be invariant to shifts and distortions (Jarrett, Kavukcuoglu, Ranzato, & LeCun, 2009; LeCun et al., 1990). Specifically, for a feature map $\mathbf{h} \in \mathbb{R}^{n*m}$ with scale (n, m), where n denotes the length of the input sequence and m denotes the number of channels, the max-pooling layer applies a sliding window of length w over each channel and results in a new feature map $\mathbf{h}'$. Each element of $\mathbf{h}'$ is given by Eq. (3).

$$\mathbf{h}'_{i,k} = max\{\mathbf{h}_{j,k}\}_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i}; \quad i = \{1, \ldots, n-w+1\}, k = \{1, \ldots, m\}. \tag{3}$$

It should be noted that an appropriate normalization could be applied to the final representations, it depends on the downstream models or tasks. We use non-normalized features by default.

3.3. Classification and clustering

TCGAN representations can, in principle, be used in many time series mining tasks. We focus on classification and clustering tasks. We will demonstrate in Section 4 that TCGAN representations are very effective in enabling linear models to achieve superior performance. It should be noted that linear models are simple and fast, but they need a nonlinear transformation of the input to process complicated data. For example, Support Vector Machine (SVM) usually applies non-linear kernel functions to make it applicable for nonlinearly separable data spaces. However, it is hard to find a suitable kernel function to improve performance, and the computation of kernels is costly and potentially infeasible. In contrast, TCGAN transformations can be learned automatically and efficiently.

For classification, we consider the following linear classifiers. Linear Support Vector machine for Classification (LSVC) and Logistic Regression(LR) are common in the machine learning community and suitable for very large datasets. SoftMax (SM) is always the last layer of a DNN for classification.

For clustering, k-means with Euclidean distance (k-means) is the most basic clustering method and assumes inputted instances can be represented as points in an Euclidean space.

Time series datasets tend to be small, and thus it is hard to split a validation set for careful hyper-parameter tuning. Therefore, we use the default settings for the above models provided in the popular machine learning toolkit scikit-learn, with the exception that SM applies a learning rate of 0.0002 and an epoch of 100.

4. Experiments

We conduct comprehensive experiments on synthetic and real time series datasets to answer the following questions:

(1) How does TCGAN learn through the adversarial game? (Section 4.2)
(2) Compared with the leading time-series GANs, how does TCGAN perform on generation and classification tasks? (Section 4.3)
(3) Compared with the superior supervised and unsupervised CNNs, how effective is TCGAN for general time series classification? (Section 4.4)
(4) How robust is the TCGAN classifier when the labeled data becomes highly limited and/or imbalanced? (Section 4.5)
(5) Are the pairwise similarities between time series well preserved in the TCGAN transformation for the clustering task? (Section 4.6)

In addition, we report TCGAN's runtime in Section 4.7. Before presenting experimental results, we describe datasets, evaluation metrics, and implementation in Section 4.1.

4.1. Experiment setup

4.1.1. Datasets

We include 85 time series datasets from the UCR repository (Chen et al., 2015) and synthetic Sines data from Yoon et al. (2019).

Following UCR repository (Chen et al., 2015), we use the default training/test splits. UCR datasets are already z-normalized. The datasets vary in type of data, the size of the training set, the size of the test set, the length of the time series and the number of classes.

- Bagnall, Bostrom, Large, and Lines (2016) categorizes the datasets into 7 types: Image Outline (29 datasets), Sensor Readings (16), Motion Capture (14), Spectrograph (7), ElectroCardioGraph (ECG) measurements (7), Electric Devices (6) and Simulated (6).
- The size of the training set is small. The number of training instances ranges from 16 (DiatomSizeReduction dataset) to 8926 (ElectricDevices dataset), with an average of 432. There are 33 datasets with sizes in the range of $[16, 100]$, 36 in the range of $[101, 500]$, and 16 in the range of $[501, 8926]$.
- In most cases, the test set is larger than the training set, ranging from 20 (BeetleFly dataset) to 8236(StarLightCurves dataset), with an average of 1164. There are 29 datasets with sizes in the range of $[20, 300]$, 32 in the range of $[301, 1000]$, and 24 in the range of $[1001, 8236]$.
- The length of the time series ranges from 24 (ItalyPowerDemand dataset) to 2709 (HandOutlines dataset), with an average of 422. There are 43 datasets with lengths in the range of $[24, 300]$, 25 datasets in the range of $[301, 700]$ and 17 datasets in the range of $[701, 2709]$.
- The number of classes ranges from 2 to 60 (ShapesAll dataset), with an average of 7. There are 71 datasets with numbers in the range of $[2, 10]$, 8 datasets in the range of $[11, 30]$, and 6 datasets in the rang of $[31, 60]$.

Following Yoon et al. (2019), we synthesize univariate and multivariate sinusoids with different frequencies and phases (Sines) for experiments. For each variable, sinusoidal sequences are sampled at different frequencies $\theta \sim Uniform(0, 1)$ and phases $\theta \sim Uniform(-\pi, \pi)$, i.e., $x = sin(2\pi\eta t + \theta)$. We consider two types of Sines data:

- SinesD1L100: includes 1-dimensional Sine time series with a length of 100.
- SinesD5L24: includes 5-dimensional Sine time series with a length of 24.

For each repeated experiment, we synthesized 10,000 samples to construct a dataset.

4.1.2. Evaluation metrics

Following existing works (Creswell et al., 2018; Esteban, Hyland, & Rätsch, 2017; Xu et al., 2020; Yoon et al., 2019), we evaluate time-series GANs by assessing the quality of the generated data, i.e., fidelity and diversity. We also consider the efficiency of GANs. To evaluate the fidelity, we visualize some generated samples and apply two metrics to quantitatively measure the similarity between real data and generated data. Specifically, for a generated dataset of the same size as the training dataset, we calculate the Nearest Neighbor Distance (NND) and Maximum Mean Discrepancy (MMD) metrics (Esteban et al., 2017; Gretton, Borgwardt, Rasch, Schölkopf, & Smola, 2007). Usually, a smaller value for both metrics signifies better GAN performance, generally the generator. Assume the real dataset is $\{\mathbf{x}^i\}_{i=1}^N$ and the generated dataset is $\{\hat{\mathbf{x}}^i\}_{i=1}^N$. MMD is defined as Eq. (4), where $K(.,.)$ is the L2-distance-based radial basis function.

$$\begin{aligned} \hat{MMD}^2 &= \frac{1}{N(N-1)}\sum_{i=1}^{N}\sum_{j\neq i}^{N} K(\mathbf{x}^i, \mathbf{x}^j) \\ &+ \frac{1}{N(N-1)}\sum_{i=1}^{N}\sum_{j\neq i}^{N} K(\hat{\mathbf{x}}^i, \hat{\mathbf{x}}^j) \\ &- \frac{2}{NN}\sum_{i=1}^{N}\sum_{j=1}^{N} K(\mathbf{x}^i, \hat{\mathbf{x}}^j) \end{aligned} \tag{4}$$

As defined in Eq. (5), NND is estimated by first calculating the distance of each generated sample from the nearest neighbor in the real dataset and then aggregating them all.

$$NND = \frac{1}{n}\sum_{i=1}^{N} min_{j=1,\ldots,N} \left\| \hat{\mathbf{x}}^i - \mathbf{x}^j \right\|^2 \tag{5}$$

To assess the diversity of generated data, we utilize t-SNE (Maaten & Hinton, 2008) to visualize how closely the distribution of the generated data matches that of the original data in 2-dimensional space. Additionally, we measure the efficiency of the GANs by calculating their training and inference time.

We specifically evaluate the usefulness of GANs for time series classification and clustering. To measure classification performance, we use the common metric, accuracy, which counts the proportion of samples that are correctly classified. For imbalanced classification scenarios, we use the weighted F1 score. A weighted F1 score is the average of binary metrics (F1 scores) weighted by the number of true instances for each class. Following Lei et al. (2019), we apply normalized mutual information (NMI) on the fused training and test splits, and set the number of classes (C) as the target number of clusters. Eq. (6) defines the NMI, where $\mathbf{U}$ and $\mathbf{V}$ are two label assignments of the same N samples in C classes, $|\mathbf{U}_i|$ and $|\mathbf{V}_j|$ are the number of samples in cluster $\mathbf{U}_i$ and $\mathbf{V}_j$. $|\mathbf{U}_i \cap \mathbf{V}_j|$ denotes the number of samples that belong to the intersection of sets $\mathbf{U}_i$ and $\mathbf{V}_j$. An NMI value close to 1 indicates high-quality clustering.

$$NMI = \frac{\sum_{i=1}^{C}\sum_{i=1}^{C} |\mathbf{U}_i \cap \mathbf{V}_j| log(\frac{N|\mathbf{U}_i \cap \mathbf{V}_j|}{|\mathbf{U}_i||\mathbf{V}_j|})}{\sqrt{(\sum_{i=1}^{C} |\mathbf{U}_i| log\frac{|\mathbf{U}_i|}{N})(\sum_{j=1}^{C} |\mathbf{V}_j| log\frac{|\mathbf{V}_j|}{N})}} \tag{6}$$

We compare two methods on multiple datasets in pairs and calculate significance using the Wilcoxon signed rank test (WSRT) with Holm correction (Garcia & Herrera, 2008; Wilcoxon, 1992). We also put multiple competitors in a Critical Difference (CD) diagram (Demšar, 2006) for compact comparison, where a thick horizontal line represents a group of approaches (a clique) that are not significantly different.

4.1.3. Implementation

We implemented our codebase in Python-3.6.13. The deep learning models were implemented with TensorFlow-2.3.0 and other machine learning models, e.g., conventional classification and clustering models, were implemented with scikit-learn-0.24.2. We ran our experiments on a server featuring 16 2.50 GHz Intel(R) Xeon(R) E5-2682 CPU cores and a NVIDIA Tesla P100 GPU. If not specified, we used a uniform set of hyper-parameters on all datasets. The settings of TCGAN were presented in Alg. 1 and Section 3. For benchmarked methods, we followed their original settings. All unsupervised procedures were trained on the fused training and test splits, and all supervised procedures were applied to the training split only. For methods with randomness, their results are reported using an average of five runs with different random seeds.

4.2. Training of TCGANs

We trained TCGAN using Alg. 1 for each dataset without any label information.

Fig. 4 shows the training and evaluation results of TCGAN on the FISH dataset. The results indicate that both the generator (G) and discriminator (D) of TCGAN do learn from the adversarial game and converge to a stable level. Specifically, Fig. 4(e) shows the competition between G and D. In the early stage, the loss of D (d_loss) drops sharply to a low level. This phenomenon signifies that D can easily learn to solve the discrimination task. Meanwhile, the increasing loss of G (g_loss) suggests that G is receiving a stronger signal from D to improve itself. As G begins to generate more realistic samples, the discrimination task of D becomes harder, and thus d_loss turns to increase. Finally, G and D reach an adversarial equilibrium. Similarly, in Fig. 4(d), both NND and MMD, which measure the similarity between generated and real samples, also converge to a relatively low level. Visually inspecting the generated time series, reveal noisy signals in the beginning (Fig. 4(a)) and appear more realistic in the end (Fig. 4(b)) becoming nearly identical with the ground truth (Fig. 4(c)).

Fig. 5 shows another case on the SinesD5L24 dataset. The results further demonstrate that TCGAN effectively learn during the adversarial game between G and D. Specifically, Fig. 5(a)–(c) are t-SNE visualizations of real data (red dots) and generated data (blue dots) at different epochs during the learning process. The real samples and generated samples move closer and closer until they perfectly overlap, indicating that the generated samples distributively cover the real data. Meanwhile, similarity measures (NND and MMD) in Fig. 5(d) consistently converge to a low level. Fig. 5(e) illustrates the healthy competition between G and D, they alternate between strength and weakness before reaching an adversarial equilibrium.

4.3. Comparisons of time-series GANs

In the following two subsections, we compare different GANs in terms of their generation performance against synthetic Sine datasets, as well as their classification performance on UCR datasets. We used the publicly available source code to implement TimeGAN (Yoon et al., 2019)[1] and CotGAN (Xu et al., 2020).[2] We followed each model's default settings with the exception of using the same number of epochs (i.e., 300) for the adversarial training. It should be noted that, prior to adversarial training, TimeGAN has to pretrain its auto-encoder component over 100 epochs and its supervised teacher-forcing component over 100 epochs. Therefore, TimeGAN actually undergoes 500 epochs in total. For more details about the competitors, please refer to Section 2.3.

4.3.1. Time-series GANs for generation

We compare TCGAN with TimeGAN and CotGAN on the SinesD1L100 and SinesD5L24 datasets. Following Yoon et al. (2019), we use a batch size of 128. After training, each GAN is used in test mode to generate a set of time series of the same size as the training set for evaluation. In Table 1 reports the mean and standard deviation of quantitative evaluation statistics over 5 random runs. We visualize samples from one single run in Figs. 6 and 7.

First, we evaluate the fidelity of generated samples of different GANs. According to the MMD and NND in Table 1, TCGAN consistently produces the best values. The results indicate that the fake samples generated by TCGAN are closer to the real samples compared with TimeGAN and CotGAN. Fig. 6 presents several generated and real samples. For the SinesD1L100 dataset, TCGAN samples appear almost identical to the real samples, while CotGAN only simulates rough contours, and TimeGAN generates flat signals significantly different from the real. For the SinesD5L24 dataset, TimeGAN samples are the closest to the real, TCGAN samples are very similar to the real but display some weak fluctuations, and CotGAN samples are significantly different from the real.

Second, t-SNE visualizations are used in Fig. 7 to assess the diversity of the generated samples. We observe that the generated samples of TCGAN show dramatically better overlap with the original samples than other GANs. In fact, in the first column, the blue (generated) samples and the red (original) samples are almost perfectly in sync. In contrast, CotGAN and TimeGAN simulate short time series (SinesD5L24 dataset) to some extent and almost fail to capture long time series (SinesD1L100 dataset).

Finally, we investigate the efficiency of different GANs. The results in Table 1 show that TCGAN is definitively the fastest model for both training and inference. On SinesD1L100 dataset, TCGAN is 14× faster in inference and 70× faster in training than the runner-up CotGAN. On SinesD5L24 dataset, TCGAN is 10× faster in inference and 38× faster in training than the runner-up TimeGAN. Furthermore, TCGAN is stable enough to achieve similar performance in both cases. In contrast, TimeGAN is slow to process long time series (SinesD1L100 datasets), and CotGAN is slow to process multi-dimensional time series (SinesD5L24 dataset).

In summary, the above results demonstrate that TCGAN efficiently generates high-quality time series by avoiding recurrence and stacking convolution layers to model entire time series.

4.3.2. Time-series GANs for classification

We evaluate the effectiveness of GANs for time series classification, using partial UCR datasets. That is 19 datasets of length up to 100, because CotGAN and TimeGAN are really time-consuming. We train GANs on the fused training and test splits in the absence of any label information, and then reuse partial pretrained modules to encode raw time series for the same off-the-shelf linear

[1] TimeGAN source code: https://github.com/jsyoon0823/TimeGAN

[2] CotGAN source code: https://github.com/tianlinxu312/cot-gan

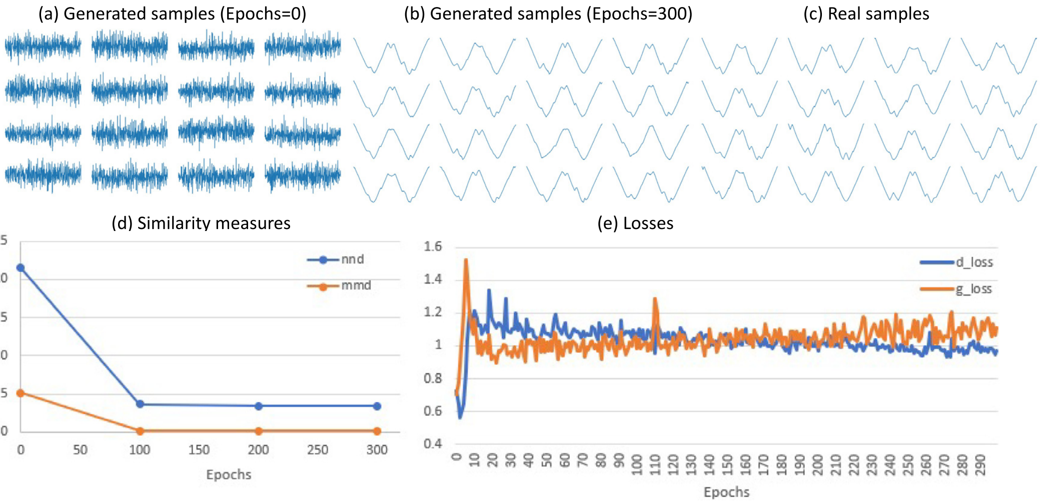


Fig. 4. Training and evaluation of TCGAN on the FISH dataset.

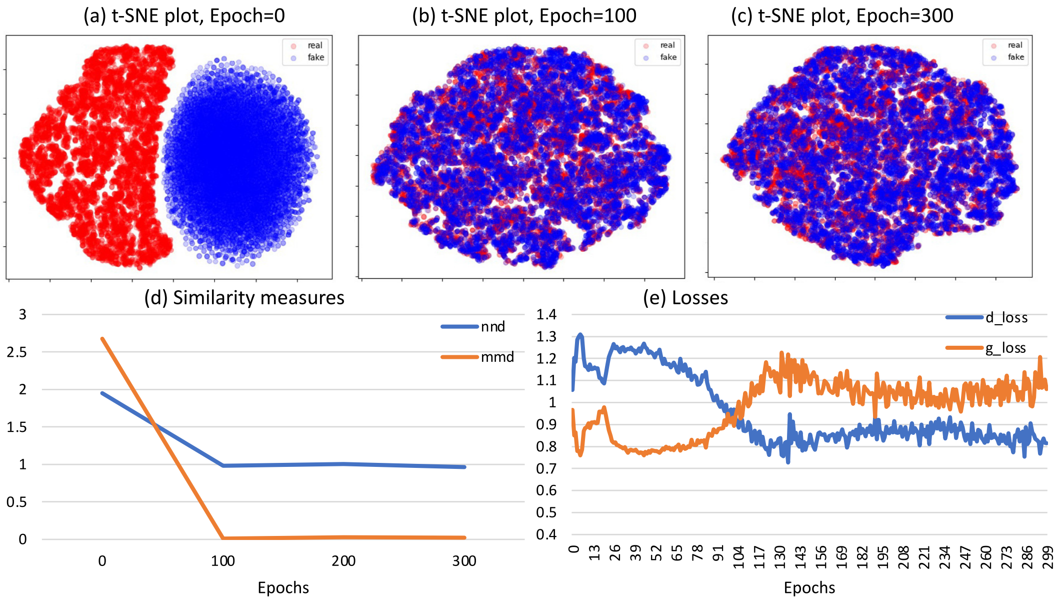


Fig. 5. Training and evaluation of TCGAN on the SinesD5L24 dataset.

Table 1
Quantitative performance of Time-series GANs on Sines data. The mean and standard deviation (in bracket) are presented for each entry. **Bold** indicates best performance.

	MMD	NND	InferTime(s)[a]	TrainTime(s)[b]
		SinesD1L100 Dataset		
TimeGAN	0.0647(0.0848)	1.4189(1.5016)	1.1682(0.1235)	63.7668(1.8127)
CotGAN	0.0222(0.0116)	1.4330(0.2610)	0.8183(0.0150)	57.6786(2.1978)
TCGAN	**0.0012(0.0008)**	**0.1112(0.0262)**	**0.0586(0.0040)**	**0.8225(0.0181)**
		SinesD5L24 Dataset		
TimeGAN	0.0244(0.0131)	1.2581(0.0740)	0.4829(0.0765)	32.6466(1.3590)
CotGAN	0.0054(0.0005)	1.4568(0.0317)	1.4432(0.1281)	103.8826(0.5975)
TCGAN	**0.0050(0.0012)**	**0.9394(0.0067)**	**0.0469(0.0011)**	**0.8593(0.0151)**

[a] The time (seconds) for generating a dataset with 10,000 samples.

[b] The time (seconds) for 1 epoch adversarial/joint training.

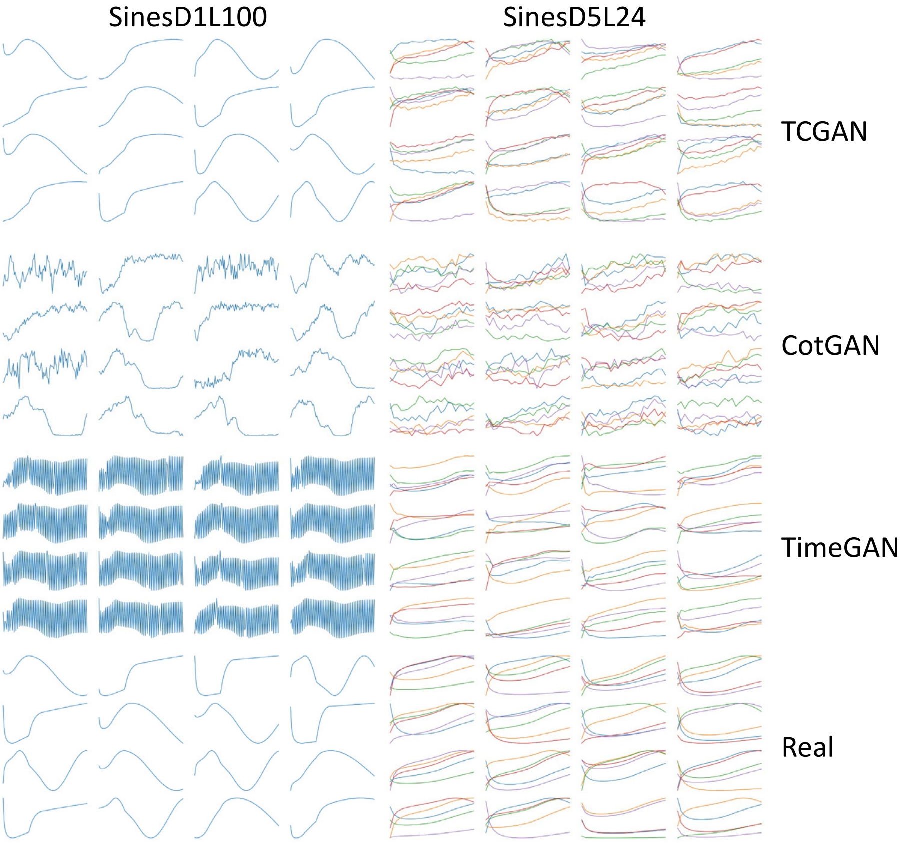


Fig. 6. Samples of SinesD1L100 (1st column) and SinesD5L24 (2nd column) obtained by different methods. Each of the top three rows presents the generated samples for each GAN. Bottom row corresponds to the real samples.

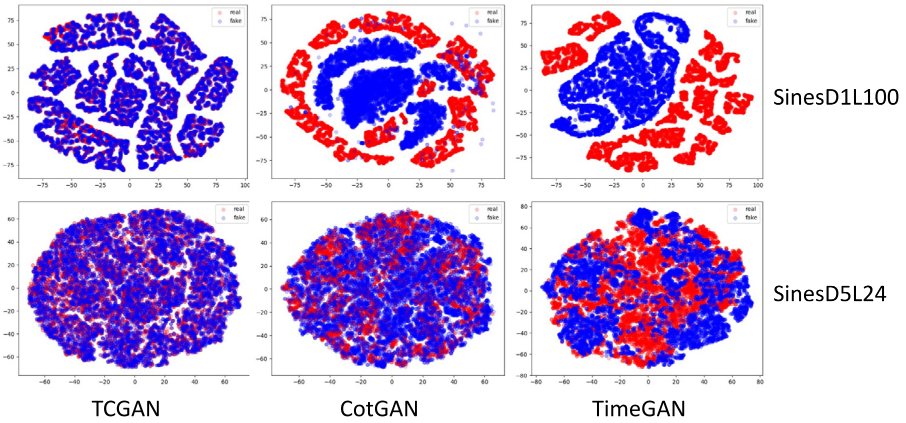


Fig. 7. t-SNE visualization on SinesD1L100 (1st row) and SinesD5L24 (2nd row). Each column corresponds to one model. Red denotes original data, and blue denotes generated data.

classifier (i.e., a softmax layer). The resulting classifiers are trained on the training split for 100 epochs. We use a small batch size (i.e., 16) because most UCR datasets contain limited samples.

The details for constructing GAN-based classifiers are as follows. We reuse TCGAN's discriminator (TCGAN) as the representation encoder. Similarly, TimeGAN has three modules that could be reused: the encoder in the auto-encoder component (TimeGAN-E), the supervised teacher-forcing component (TimeGAN-S) and the discriminator (TimeGAN-D). CotGAN has two discriminators, D_h and D_M jointly approximate the optimization objective. Therefore, we reuse D_h and D_M to form CotGAN-Dh and CotGAN-DM, respectively. For each of the above representation encoders, we flatten the outputs from the last hidden layer and feed them into a softmax layer to conduct classification.

Table 2 illustrates the results. For TimeGAN and CotGAN that derive multiple encoders or classifiers, we report the best accuracy of all options. The results show that TCGAN significantly outperforms the competitors and wins 15 out of 19 datasets. We

Table 2
GANs for time series classification on 19 UCR datasets of length up to 100. Each entry corresponds to the mean and standard deviation (in bracket) of accuracies. **Bold** indicates best performance.

Dataset	CotGAN	TimeGAN	TCGAN
DistalPhalanxOutlineAgeGroup	0.8066 (0.0011)	0.7925 (0.0045)	**0.8265 (0.0471)**
DistalPhalanxOutlineCorrect	0.6137 (0.0022)	0.6857 (0.0189)	**0.8183 (0.0136)**
DistalPhalanxTW	0.7369 (0.0011)	0.7797 (0.0027)	**0.79 (0.0085)**
ECG200	0.8132 (0.0066)	0.8196 (0.0068)	**0.926 (0.0089)**
ElectricDevices	0.2424 (0.0363)	0.6072 (0.0030)	**0.628 (0.027)**
ItalyPowerDemand	0.7909 (0.0348)	0.9630 (0.0015)	**0.9679 (0.0036)**
MedicalImages	0.5175 (0.0157)	0.6304 (0.0132)	**0.7676 (0.0131)**
MiddlePhalanxOutlineAgeGroup	0.2845 (0.0024)	**0.7919 (0.0062)**	0.779 (0.0118)
MiddlePhalanxOutlineCorrect	0.5279 (0.0013)	0.5345 (0.0039)	**0.716 (0.0999)**
MiddlePhalanxTW	0.4566 (0.0641)	**0.6418 (0.0021)**	0.6226 (0.012)
MoteStrain	0.5428 (0.1236)	**0.8711 (0.0033)**	0.8246 (0.0114)
PhalangesOutlinesCorrect	0.6689 (0.0143)	0.6924 (0.0067)	**0.7953 (0.0108)**
ProximalPhalanxOutlineAgeGroup	0.5065 (0.0132)	**0.8500 (0.0032)**	0.8098 (0.0329)
ProximalPhalanxOutlineCorrect	0.6797 (0.0)	0.8080 (0.0137)	**0.8577 (0.011)**
ProximalPhalanxTW	0.6721 (0.1225)	0.8008 (0.0018)	**0.805 (0.0116)**
SonyAIBORobotSurface	0.6202 (0.0117)	0.6848 (0.0055)	**0.8722 (0.0398)**
SonyAIBORobotSurfaceII	0.7835 (0.0104)	0.8263 (0.0049)	**0.9158 (0.0095)**
synthetic_control	0.6411 (0.0079)	0.8943 (0.0114)	**0.9913 (0.0069)**
TwoLeadECG	0.699 (0.0633)	0.9476 (0.0025)	**0.9867 (0.0079)**
Win	0	4	15

attribute this success to the stacking convolution layers in TCGAN that can model contextual and hierarchical features inherent in raw time series. In contrast, CotGAN and TimeGAN depend on auto-regressive modules that prefer to attend to values in the recent past, and the accumulation process may have the vanishing issue and accumulation errors. We notice that the authors of TimeGAN usually use a window size of 24 to slice the time series. We also observe that TimeGAN outperforms CotGAN on almost all datasets (18 datasets). However, the results presented in the previous section (see Table 1) show the generation performance of CotGAN is close to or slightly better than TimeGAN. This phenomenon suggests that pursuing strong performance on the generation task, which is the focus of existing time-series GANs, does not definitely bring benefits to the discrimination task.

4.4. TCGAN representations for classification

As described in Section 3.3, we feed TCGAN representations to linear classifiers for time series classification. Using Linear Support Vector Machine for classification (LSVC), Logistic Regression (LR) and SoftMax layer (SM), we have **LSVC-TCGAN**, **LR-TCGAN** and **SM-TCGAN** classifiers, respectively.

For fair comparison, we use TCGAN as the backbone to construct the following CNN classifiers:

- **Supervised CNNs**. We replace the sigmoid output layer of TCGAN discriminator with a softmax layer to construct a standard FCN for supervised multi-class classification. We train the resulting model from scratch (**TCGAN-D**) or only randomly initialize it (**TCGAN-D-R**). We also include the **FCN** proposed in Fawaz et al. (2019).
- **Unsupervised CNNs**. GANs are innately similar to AutoEncoders (AEs) that have been widely used in unsupervised learning (Kingma & Welling, 2019). We reform TCGAN to construct an AE structure where the discriminator and generator of TCGAN are used as the encoder and decoder of AE. Similarly, we froze the pretrained AE encoder and connect it with a SoftMax layer for classification (**SM-AE**) as the same as SM-TCGAN. We also consider advanced variants of AE: denoising AE (DAE) and Variational AE (VAE). Similar to SM-AE, we build **SM-DAE** and **SM-VAE** classifiers. In the case of VAE, we reuse the layer before the middle bottleneck layer because we have the same observation as Zhang, Yao, and Yuan (2019) that the latent representations output from the bottleneck layer are randomly sampled from a multivariate Gaussian distribution and thus are inappropriate for classification. In fact, the resulting VAE corresponds to the VAE++ proposed in Zhang et al. (2019).

We also include recent SOTA representation learning methods for time series classification.

- Eldele et al. (2022) extends Time-Series representation learning framework via Temporal and Contextual Contrasting (**TS-TCC**) to propose Class-Aware TS-TCC (**CA-TCC**) for semi-supervised time series classification. TS-TCC and CA-TCC are CNNs and use an unsupervised-pretrained encoder for a linear classifier. Specifically, TS-TCC learns representations from unlabeled data with contrastive learning on time-series-specific weak and strong augmentations. TS-TCC has a temporal contrasting module to capture temporal relationships and a contextual contrasting module to capture discriminative representations. The self-supervised pretrained TS-TCC is then used as a representation encoder to collaborate with a linear classifier (i.e., a softmax layer) for time series classification. The resulting classifier has two settings: (1) **TS-TCC-L(inear)** frozen the encoder and (2) **TS-TCC-T(une)** fine-tune the encoder in the supervised training process. Based on TS-TCC, CA-TCC adds two phases to utilize labeled data in the supervised learning process. CA-TCC deploys TS-TCC-T, which has been fine-tuned on (limited) real-labeled data, to generate pseudo labels for the entire unlabeled dataset. Then, the unsupervised contextual contrastive module of TS-TCC is replaced with a supervised contextual contrasting module in the CA-TCC to ingest the pseudo labels. We use the open-source code[3] to reproduce TS-TCC-L, TS-TCC-T, and CA-TCC.
- TS2Vec is an unsupervised representation learning framework for time series that has demonstrated superior performance in the classification task (Yue et al., 2021). It employs a Temporal Convolutional Network (TCN) as backbone, incorporating dilated convolutions for time series forecasting. TS2vec performs contrastive learning in a hierarchical manner, considering augmented context views to capture multi-scale features of time series at both the timestamp and the instance levels. Additionally, a random cropping strategy is adopted to generate new contexts for

[3] TS-TCC and CA-TCC: https://github.com/emadeldeen24/CA-TCC

learning position-agnostic representations. To utilize the pretrained representations for classification, the authors employ a non-linear classifier, a SVM with RBF kernel, and selected the penalty C by grid search. We refer to this classifier as **SVM-TS2Vec**. For fair comparison, we also connect the pretrained TS2Vec with the linear classifiers SM, LSVC and LR and derive **SM-TS2Vec**, **LSVC-TS2Vec** and **LR-TS2Vec**, respectively. We use the open-source code[4] to reproduce the TS2Vec and its associated classifiers.

We run experiments on 85 UCR datasets for five repetitions. For a compact comparison, we put the results in a critical difference diagram (Fig. 8) and have the following observations.

- TCGAN classifiers consistently rank in the top group. The results manifest that TCGAN can learn useful representations in an unsupervised manner to boost simple classifiers to achieve superior performance.
- TCGAN-D-R is absolutely the worst model. Therefore, the superior performance of the above TCGAN classifiers does not come from random convolutions.
- SM-TCGAN outperforms TCGAN-D by a large margin. This result indicates that, in comparison with training a supervised network from scratch, TCGAN can improve classification performance by making use of additional unlabeled data (samples in test split without label information) for unsupervised pretraining. In addition, TCGAN-D is close to the FCN that verifies the correctness of SM-TCGAN and other variants.
- The unsupervised networks (SM-AE, SM-DAE and SM-VAE) are inferior to the supervised networks (FCN and TCGAN-D). This phenomenon aligns with the conclusions in Fawaz et al. (2019). Again, the results demonstrate the superiority of TCGAN by significantly outperforming peer supervised networks via the same unsupervised learning schema as AEs.
- TS-TCC-L is close to the AE variants, while TS-TCC-T is competitive with LSVC-TSGAN. This phenomenon suggests that the representations learned in the pretrained TS-TCC are not well aligned with the real labels, and thus fine-tuning is required to break the gap for better performance. In comparison, the frozen pretrained TCGAN works well with the linear classifiers. CA-TCC is better than TS-TCC-L with the benefit of fine-tuning, while it is worse than TS-TCC-T, indicating that pseudo labels can have negative effects. In fact, CA-TCC is designed for semi-supervised learning with limited real-labeled data. We will have a further discussion in Section 4.5.1.
- SVM-TS2Vec achieves the highest accuracy, confirming the conclusions drawn in the original paper that emphasize the benefits of fine-grained and multi-scale features. The other models do not encapsulate representations in different levels of granularity like TS2Vec. However, we observed that the pretrained TS2Vec representations are not well-suited for linear classifiers. LSVC-TS2Vec, LR-TS2Vec and SM-TS2Vec perform worse than the TCGAN classifiers. SM-TS2Vec is even the worst classifier except the randomly initialized model TCGAN-D-R. This observation suggests that achieving superior performance with TS2Vec representations requires a strong classifier and careful fine-tuning process. Furthermore, in Section 4.5.1, we will demonstrate that TS2Vec is generally less effective than TCGAN in the semi-supervised scenario.

The above results demonstrate that TCGAN does automatically learn a good representation of raw data that makes the classification task easier.

To investigate the high variance problem mentioned in Fig. 1, we compare SM-TCGAN with FCN on 18 UCR datasets over which FCNs have standard deviations greater than 0.1. TCGAN wins 15 datasets, and all results have dramatically smaller standard deviations. In average, SM-TCGAN (0.8188 ± 0.0259) is significantly more accurate and stable than FCN (0.6440 ± 0.1657). The result suggests that TCGAN shows promise in alleviating the high variance problem widely prevalent in supervised networks. We attribute this improvement to the unsupervised pretraining network in SM-TCGAN. In fact, joint training the layers of a supervised DNN is difficult, and the network can easily converge to a bad local minimum with an inappropriate initialization. As a result, the performance tends to be unstable across multiple runs with different random initializations, particularly for small data. Existing works (Erhan, Courville, Bengio, & Vincent, 2010) have demonstrated that unsupervised pretraining has a regularization effect and can guide the learning towards basins of attraction of minima that support better generalization from the training dataset (see Table 3).

4.5. TCGAN representations for classification in extreme situations

4.5.1. TCGAN representations for classification in absence of labeled data

First, we compare LR-TCGAN with advanced single-supervised CNNs, i.e., the FCN and ResNet in Fawaz et al. (2019). We consider the FordA and wafer datasets because both datasets are large enough and all classifiers obtain competitive accuracy on the complete training set. We run each setting 5 times with different random seeds and report the average accuracy on the default test split. Fig. 9 presents the results. A subplot corresponds to one dataset, x-axis denotes the proportion of labeled training data used for supervised learning, and y-axis is the average accuracy. As expected, with a decreasing proportion of available labels, all classifiers consistently get worse, but the decay of fully supervised networks (ResNet and FCN) is significantly greater than that of TCGAN classifiers with unsupervised pretraining. Moreover, ResNet and FCN tend to be unstable on the imbalanced dataset wafer.

Moreover, we compare LR-TCGAN with two leading semi-supervised models: **CA-TCC** (Eldele et al., 2022) and Semi-supervised Time series classification architecture (**SemiTime**) (Fan, Zhang, Wang, Huang, & Li, 2021). CA-TCC has been introduced in Section 4.4. SemiTime ingests labeled and unlabeled data simultaneously. Specifically, SemiTime conducts supervised classification directly on the labeled data. To utilize the unlabeled data, SemiTime proposes a self-supervised predictor that samples segments of past-future pair from time series and predicts the temporal relation between them. We reproduce SemiTime using the open source code.[5] Additionally, we evaluate the top two classifiers associated with TS2Vec (Yue et al., 2021), i.e., **SVM-TS2Vec** and **LSVC-TS2Vec**, in the semi-supervised scenario. Please refer to Section 4.4 for more details on TS2Vec.

Our experiments encompass 24 UCR datasets. Specifically, we rank the 85 UCR datasets by the average number of samples in each class and select the top 15 datasets. Besides, we include 9 datasets from competitors (Eldele et al., 2022; Fan et al., 2021).

We train semi-supervised models with different proportions of labeled training data and report the accuracy on the test data. Tables 4–6 present the results, with each entry representing the average accuracy over 5 random runs. TCGAN is generally

[4] TS2Vec: https://github.com/yuezhihan/ts2vec

[5] SemiTime: https://github.com/haoyfan/SemiTime

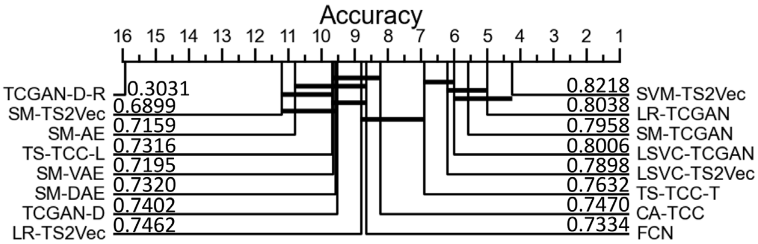


Fig. 8. Critical difference diagram for CNN classifiers on 85 UCR datasets. The float numbers are the average accuracies.

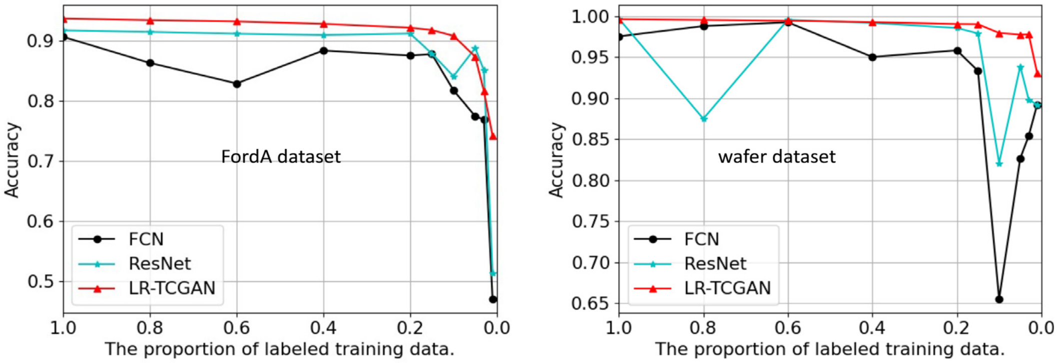


Fig. 9. Classification accuracy varies with respect to the size of labeled training dataset.

Table 3
Mean and standard deviation (Std) of classification accuracies. On the selected 18 UCR datasets, FCNs have standard deviations greater than 0.1.

Dataset	Mean		Std	
	FCN	TCGAN	FCN	TCGAN
ArrowHead	0.672	**0.8514**	0.1973	**0.0114**
Beef	0.4733	**0.84**	0.1321	**0.0279**
BirdChicken	**0.88**	0.69	0.1151	**0.0894**
Car	0.6667	**0.88**	0.172	**0.0139**
CBF	0.9271	**0.9964**	0.1475	**0.0009**
ChlorineConcentration	0.4661	**0.8837**	0.1646	**0.0067**
CinC_ECG_torso	0.6351	**0.9722**	0.1181	**0.0236**
Cricket_Y	0.6764	**0.7287**	0.11	**0.0201**
DistalPhalanxOutlineCorrect	0.6757	**0.8183**	0.1311	**0.0136**
Earthquakes	0.5193	**0.7752**	0.2256	**0.0135**
FaceFour	0.7523	**0.9091**	0.2582	**0.018**
Gun_Point	0.9253	**0.9707**	0.1081	**0.0174**
MALLAT	0.6641	**0.9431**	0.2154	**0.0118**
Meat	0.43	**0.8967**	0.166	**0.0075**
MiddlePhalanxOutlineCorrect	**0.74**	0.716	0.1035	**0.0999**
OliveOil	0.2133	**0.5067**	0.107	**0.0596**
OSULeaf	**0.7802**	0.6769	0.3102	**0.0125**
SmallKitchenAppliances	0.4949	**0.6837**	0.2011	**0.0178**
Avg	0.6440	**0.8188**	0.1657	**0.0259**

the best. Specifically, TCGAN wins 13, 13, and 14 out of 24 datasets with respect to 40%, 20%, and 10% of labeled training data, respectively.

In summary, the above results demonstrate that TCGAN classifiers can make use of unlabeled data to obtain a decent and stable accuracy even when the labeled data are highly limited.

4.5.2. TCGAN representations for imbalanced classification

We explore the effectiveness of LR-TCGAN and advanced single-supervised CNNs (i.e., the FCN and ResNet in Fawaz et al., 2019) for imbalanced classification. We include 5 UCR datasets with imbalance ratios greater than 30. ImBalance Ratio (IBRa) is the ratio between the number of samples from the most and least frequent classes. We use the common imbalanced classification evaluation metric, the weighted F1 score. The results in Table 7 show that LR-TCGAN achieves the best results on all datasets.

4.6. TCGAN representations metric space for clustering

Effective clustering requires a representation space that defines a meaningful measure between different samples and uncovers discriminative structures inherent in the data. We will demonstrate that TCGAN transformation can enable a simple clustering method to achieve superior performance.

Table 4
Classification accuracies of semi-supervised methods with 40% of labeled training set. **Bold** indicates best performance.

Dataset	TCGAN	CA-TCC	SemiTime	SVM-TS2Vec	LSVC-TS2Vec
ChlorineConcentration	**0.7314**	0.5369	0.6242	0.5988	0.566
Cricket_X	0.6251	0.5687	0.5271	**0.6862**	0.6785
DistalPhalanxOutlineCorrect	**0.8103**	0.7453	0.7433	0.7406	0.7399
ElectricDevices	0.6354	0.6752	0.306	**0.6988**	0.6743
FordA	0.9008	0.9117	0.8905	**0.9315**	0.9182
FordB	**0.8958**	0.895	0.8776	0.7933	0.7775
HandOutlines	0.8554	0.8354	0.6544	0.9103	**0.9141**
InsectWingbeatSound	0.5738	0.5911	0.4512	0.5715	**0.6043**
MiddlePhalanxOutlineCorrect	0.593	0.5233	0.6734	**0.7808**	0.6357
NonInvasiveFatalECG_Thorax1	**0.9138**	0.829	0.8263	0.8861	0.8328
NonInvasiveFatalECG_Thorax2	**0.9227**	0.855	0.845	0.9044	0.8673
PhalangesOutlinesCorrect	**0.7986**	0.6704	0.7345	0.7869	0.7443
ProximalPhalanxOutlineAgeGroup	0.8507	0.8166	0.7461	0.8341	**0.8693**
ProximalPhalanxOutlineCorrect	**0.8454**	0.754	0.7839	0.8419	0.7904
ShapesAll	0.715	0.5553	0.7084	0.808	**0.8093**
StarLightCurves	**0.9635**	0.9509	0.9573	0.963	0.9588
Strawberry	**0.9602**	0.6936	0.9161	0.9508	0.8957
Two_Patterns	0.9552	**0.9971**	0.9944	0.9964	0.993
UWaveGestureLibraryAll	**0.951**	0.9277	0.7862	0.882	0.8778
uWaveGestureLibrary_X	**0.8202**	0.7535	0.7136	0.7691	0.7636
uWaveGestureLibrary_Y	**0.7095**	0.6601	0.5868	0.6557	0.649
uWaveGestureLibrary_Z	**0.7318**	0.6639	0.6643	0.708	0.7006
wafer	0.9921	0.9882	0.9774	**0.9966**	0.995
yoga	0.7841	0.6757	0.6822	**0.8077**	0.7575
Avg	**0.8140**	0.7531	0.7363	0.8126	0.7922
Wins	**13**	1	0	6	4

Table 5
Classification accuracies of semi-supervised methods with 20% of labeled training set. **Bold** indicates best performance.

Dataset	TCGAN	CA-TCC	SemiTime	SVM-TS2Vec	LSVC-TS2Vec
ChlorineConcentration	**0.6041**	0.5068	0.5396	0.5483	0.5517
Cricket_X	0.5297	0.4826	0.4565	0.5954	**0.6077**
DistalPhalanxOutlineCorrect	**0.759**	0.6773	0.7084	0.7464	0.7217
ElectricDevices	0.6187	0.6662	0.3016	**0.6859**	0.6632
FordA	0.891	0.8931	0.8937	**0.925**	0.9185
FordB	**0.8829**	0.8547	0.8769	0.7798	0.778
HandOutlines	0.841	0.8246	0.699	0.8941	**0.907**
InsectWingbeatSound	0.5241	0.5426	0.3514	0.5044	**0.5525**
MiddlePhalanxOutlineCorrect	0.6567	0.5033	0.6113	**0.7808**	0.5876
NonInvasiveFatalECG_Thorax1	**0.8675**	0.7264	0.7674	0.8295	0.7829
NonInvasiveFatalECG_Thorax2	**0.8925**	0.7563	0.7955	0.8663	0.8315
PhalangesOutlinesCorrect	**0.7737**	0.634	0.7357	0.7555	0.6991
ProximalPhalanxOutlineAgeGroup	0.8527	0.8078	0.7216	0.8351	**0.8585**
ProximalPhalanxOutlineCorrect	**0.8378**	0.7189	0.753	0.8144	0.7196
ShapesAll	0.6113	0.6097	0.5534	0.7203	**0.728**
StarLightCurves	0.9431	0.8853	**0.965**	0.9491	0.9429
Strawberry	**0.9527**	0.6943	0.8948	0.8784	0.8573
Two_Patterns	0.8796	0.9546	0.9673	**0.9902**	0.9806
UWaveGestureLibraryAll	**0.9322**	0.8958	0.7358	0.8306	0.832
uWaveGestureLibrary_X	**0.7891**	0.7059	0.6725	0.7178	0.7296
uWaveGestureLibrary_Y	**0.6773**	0.6132	0.5455	0.6056	0.6013
uWaveGestureLibrary_Z	**0.7001**	0.6267	0.6376	0.6788	0.6662
wafer	**0.987**	0.9726	0.9639	0.9865	0.986
yoga	0.712	0.6457	0.6355	**0.7162**	0.6899
Avg	**0.7798**	0.7166	0.6993	0.7764	0.7581
Wins	**13**	0	1	5	5

We make comparisons with the following models:

- **k-means** with Euclidean distance and raw data input is a common clustering method.
- **k-means-TCGAN**. Our model feeds TCGAN representations to a k-means with Euclidean distance.
- **k-shape** (Paparrizos & Gravano, 2017) is a superior raw-data-based algorithm that uses a normalized version of the cross-correlation as a distance measure to cluster raw data.
- **k-means-SPIRAL-DTW** (Lei et al., 2019) is an advanced feature-based method. It trains a k-means algorithm with DTW distance on the extracted SPIRAL features.

We implement k-means with scikit-learn, run k-shape with the public source code,[6] and directly copy the result of k-means-SPIRAL-DTW from the primitive paper due to unavailability of the source code. All experiments are run 5 times with different random seeds. The k-means with raw data input is deterministic and thus is run once.

Table 8 reports the average of clustering NMIs on all UCR 85 datasets. Our method, k-means-TCGAN, achieves the best results, even though k-means-SPIRAL-DTW applies a more complicated distance measure, DTW. Furthermore, we observe the pairwise

[6] k-shape source code: https://tslearn.readthedocs.io/en/stable/gen_modules/clustering/tslearn.clustering.KShape.html.

Table 6
Classification accuracies of semi-supervised methods with 10% of labeled training set. **Bold** indicates best performance.

Dataset	TCGAN	CA-TCC	SemiTime	SVM-TS2Vec	LSVC-TS2Vec
ChlorineConcentration	**0.5524**	0.484	0.4916	0.4882	0.5385
Cricket_X	0.4374	0.3677	0.401	0.4482	**0.4836**
DistalPhalanxOutlineCorrect	0.6587	0.6433	0.664	**0.7246**	0.6906
ElectricDevices	0.5915	0.6405	0.2915	**0.6667**	0.647
FordA	0.8695	0.8764	0.8819	**0.9197**	0.905
FordB	**0.8763**	0.8149	0.832	0.782	0.7728
HandOutlines	0.8186	0.8176	0.6068	0.8935	**0.9043**
InsectWingbeatSound	**0.483**	0.4809	0.2772	0.4444	0.4738
MiddlePhalanxOutlineCorrect	0.6147	0.4713	0.5918	**0.6948**	0.5581
NonInvasiveFatalECG_Thorax1	**0.8063**	0.5976	0.6807	0.7421	0.7063
NonInvasiveFatalECG_Thorax2	**0.8417**	0.6559	0.7113	0.8004	0.7702
PhalangesOutlinesCorrect	**0.7364**	0.6179	0.6832	0.7354	0.6685
ProximalPhalanxOutlineAgeGroup	0.8371	0.8107	0.7312	0.7951	**0.8478**
ProximalPhalanxOutlineCorrect	**0.8027**	0.7285	0.6918	0.7677	0.7065
ShapesAll	0.4957	0.497	0.4037	0.6177	**0.6377**
StarLightCurves	0.9373	0.9004	**0.973**	0.9346	0.9236
Strawberry	**0.9044**	0.6998	0.8719	0.8578	0.82
Two_Patterns	0.7892	0.9002	0.9001	**0.9654**	0.954
UWaveGestureLibraryAll	**0.891**	0.8784	0.6943	0.7513	0.7639
uWaveGestureLibrary_X	**0.751**	0.6726	0.6126	0.6759	0.6838
uWaveGestureLibrary_Y	**0.627**	0.5973	0.4734	0.5598	0.5512
uWaveGestureLibrary_Z	**0.6473**	0.5976	0.5887	0.6442	0.64
wafer	**0.9731**	0.9687	0.9592	0.9722	0.9687
yoga	**0.6357**	0.5937	0.5797	0.6279	0.6058
Avg	**0.7324**	0.6797	0.6497	0.7296	0.7176
Wins	**14**	0	1	5	4

Table 7
Mean and standard deviation (in bracket) of weighted F1 scores on imbalanced datasets. **Bold** indicates best performance.

Dataset	IBRa	FCN	ResNet	LR-TCGAN
50words	52.000	0.5850(0.0146)	0.7054(0.0228)	**0.7592(0.0094)**
ECG5000	146.000	0.9300(0.0065)	0.9281(0.0037)	**0.9324(0.0010)**
MedicalImages	33.833	0.7211(0.0718)	0.7625(0.0143)	**0.7699(0.0144)**
ProximalPhalanxTW	36.000	0.7565(0.0439)	0.7682(0.0118)	**0.7825(0.0124)**
WordsSynonyms	30.000	0.4683(0.0510)	0.5872(0.0173)	**0.6650(0.0117)**

Table 8
Average of clustering NMIs on 85 UCR datasets. **Bold** indicates best performance.

	k-means-TCGAN	k-means	k-shape	k-means-SPIRAL-DTW
NMI	**0.3412**	0.2889	0.3097	0.332

Table 9
Pairwise comparison of clustering methods on 85 UCR datasets.

	>	=	<	p(WSRT)
k-means-TCGAN vs. k-means	64	2	19	5.64E−07
k-means-TCGAN vs. k-shape	51	0	34	0.0149
k-shape vs. k-means	48	0	37	0.1551

comparison results of k-means-TCGAN, k-means and k-shape. k-means-SPIRAL-DTW is excluded because the detailed results for each dataset are not available. In Table 9, columns ">", "=", "<" denote the number of datasets over which the active competitor is better, equal, or worse, respectively, in comparison with the other. Column "p(WSRT)" presents the p-values of the Wilcoxon signed rank test (WSRT). The results show that k-means-TCGAN is significantly better than k-means and k-shape ($p(\text{WSRT}) < 0.05$). It should be noted that the k-means algorithm in scikit-learn uses k-means++ for initialization, while k-shape uses an inferior initialization method (random initialization). Therefore, in our experiments, k-shape does not significantly outperform k-means. In fact, it is not trivial to use k-means++ in k-shape. From this perspective, an independent encoder is more flexible for use in conjunction with many advanced clustering algorithms.

We also apply t-SNE (Maaten & Hinton, 2008) to find a two-dimensional embedding and inspect the representation space. Fig. 2 visualizes the Two_Patterns dataset where instances of different classes become more distinguishable in the representation space. As a side effect, such a t-SNE plot could be an exploratory data analysis technique to help domain experts understand time series as in Yeh, Van Herle, and Keogh (2016).

4.7. Runtime

We report the total runtime over 85 UCR datasets. Training TCGAN takes 8.6834 h. Extracting features requires 38.9938 s. SM-TCGAN, LR-TCGAN and LSVC-TCGAN classifiers spend 2.9955 h, 0.1111 h, and 0.0913 h, respectively. Clustering with kmeans takes 0.1332 h. In addition, we have discussed the runtime of time-series GANs in Section 4.3, particularly Table 1. Similar to existing DNNs, training neural networks (i.e., TCGAN and SM classifier) with enough epochs takes the most time. It is reassuring that TCGAN only needs to be trained once for all downstream applications.

5. Conclusion

We introduced TCGAN for time series classification and clustering. TCGAN is trained by playing an adversarial game in absence any labeled information, and then parts of the trained TCGAN are reused to construct a representation encoder for linear classification and clustering methods. Our extensive experiments on synthetic and real-world time series datasets demonstrate that TCGAN outperforms existing time-series GANs in terms of effectiveness and efficiency. TCGAN representations enable simple classifiers to achieve higher accuracy and stability than both leading unsupervised and supervised CNNs even in highly limited and/or imbalanced labeled data scenarios. Plus, the pairwise similarities between time series are well preserved in TCGAN transformation, making the distance-based clustering method more effective. We hope our work will motivate further research on applying GANs or generative models to address the shortage of labeled time series data.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

I have shared the link to my data and code.

References

Aghabozorgi, S., Shirkhorshidi, A. S., & Wah, T. Y. (2015). Time-series clustering–a decade review. *Information Systems*, *53*, 16–38.

Bagnall, A., Bostrom, A., Large, J., & Lines, J. (2016). The great time series classification bake off: An experimental evaluation of recently proposed algorithms. extended version. arXiv preprint arXiv:1602.01711.

Bagnall, A., Lines, J., Bostrom, A., Large, J., & Keogh, E. (2017). The great time series classification bake off: A review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, *31*(3), 606–660.

Bai, S., Kolter, J. Z., & Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv preprint arXiv:1803.01271.

Brophy, E., Wang, Z., She, Q., & Ward, T. (2021). Generative adversarial networks in time series: A survey and taxonomy. arXiv preprint arXiv:2107.11098.

Chen, Y., Keogh, E., Hu, B., Begum, N., Bagnall, A., Mueen, A., et al. (2015). The UCR time series classification archive. www.cs.ucr.edu/~eamonn/time_series_data/.

Christ, M., Braun, N., Neuffer, J., & Kempa-Liehr, A. W. (2018). Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing*, *307*, 72–77.

Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., & Bharath, A. (2018). Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, *35*, 53–65.

Cui, Q., Sun, H., Kong, Y., Zhang, X., & Li, Y. (2021). Efficient human motion prediction using temporal convolutional generative adversarial network. *Information Sciences*, *545*, 427–447.

Dahl, C. M., & Sørensen, E. N. (2021). Time series (re)sampling using generative adversarial networks. *Neural Networks : The Official Journal of the International Neural Network Society*, *156*, 95–107.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*, 1–30.

Doersch, C., & Zisserman, A. (2017). Multi-task self-supervised visual learning. In *Proceedings of the IEEE international conference on computer vision* (pp. 2051–2060).

Eldele, E., Ragab, M., Chen, Z., Wu, M., Kwoh, C.-K., Li, X., et al. (2022). Self-supervised contrastive representation learning for semi-supervised time-series classification. arXiv preprint arXiv:2208.06616.

Erhan, D., Courville, A., Bengio, Y., & Vincent, P. (2010). Why does unsupervised pre-training help deep learning? In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 201–208). JMLR Workshop and Conference Proceedings.

Esteban, C., Hyland, S. L., & Rätsch, G. (2017). Real-valued (medical) time series generation with recurrent conditional gans. arXiv preprint arXiv:1706.02633.

Fan, H., Zhang, F., Wang, R., Huang, X., & Li, Z. (2021). Semi-supervised time series classification by temporal relation prediction. In *ICASSP 2021-2021 IEEE international conference on acoustics, speech and signal processing* (pp. 3545–3549). IEEE.

Fawaz, H. I., Forestier, G., Weber, J., Idoumghar, L., & Muller, P.-A. (2019). Deep learning for time series classification: A review. *Data Mining and Knowledge Discovery*, *33*(4), 917–963.

Garcia, S., & Herrera, F. (2008). An extension on" statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons.. *Journal of Machine Learning Research*, *9*(12).

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., et al. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672–2680).

Gretton, A., Borgwardt, K., Rasch, M., Schölkopf, B., & Smola, A. J. (2007). A kernel method for the two-sample-problem. In *Advances in neural information processing systems* (pp. 513–520).

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. C. (2017). Improved training of wasserstein gans. In *Advances in neural information processing systems* (pp. 5767–5777).

Hills, J., Lines, J., Baranauskas, E., Mapp, J., & Bagnall, A. (2014). Classification of time series by shapelet transformation. *Data Mining and Knowledge Discovery*, *28*(4), 851–881.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

Ismail Fawaz, H., Lucas, B., Forestier, G., Pelletier, C., Schmidt, D. F., Weber, J., et al. (2020). Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery*, *34*(6), 1936–1962.

Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. ArXiv Preprint.

Jarrett, D., Bica, I., & van der Schaar, M. (2021). Time-series generation by contrastive imitation. *Advances in Neural Information Processing Systems*, *34*.

Jarrett, K., Kavukcuoglu, K., Ranzato, M., & LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th international conference on computer vision* (pp. 2146–2153). IEEE.

Javed, A., Lee, B. S., & Rizzo, D. M. (2020). A benchmark study on time series clustering. *Machine Learning with Applications*, *1*, Article 100001.

Jenni, S., & Favaro, P. (2018). Self-supervised feature learning by learning to spot artifacts. In *2018 IEEE/CVF conference on computer vision and pattern recognition* (pp. 2733–2742).

Kashiparekh, K., Narwariya, J., Malhotra, P., Vig, L., & Shroff, G. (2019). Convtimenet: A pre-trained deep convolutional neural network for time series classification. In *2019 international joint conference on neural networks* (pp. 1–8). IEEE.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kingma, D. P., & Welling, M. (2019). An introduction to variational autoencoders. arXiv preprint arXiv:1906.02691.

Längkvist, M., Karlsson, L., & Loutfi, A. (2014). A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, *42*, 11–24.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436.

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., et al. (1990). Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems* (pp. 396–404).

Lei, Q., Yi, J., Vaculin, R., Wu, L., & Dhillon, I. S. (2019). Similarity preserving representation learning for time series clustering. In *IJCAI'19* (pp. 2845–2851).

Li, D., Chen, D., Goh, J., & Ng, S.-K. (2018). Anomaly detection with generative adversarial networks for multivariate time series. arXiv preprint arXiv:1809.04758.

Lines, J., & Bagnall, A. (2015). Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, *29*(3), 565–592.

Luo, Y., Zhang, Y., Cai, X., & Yuan, X. (2019). E2gan: End-to-end generative adversarial network for multivariate time series imputation. In *Proceedings of the 28th international joint conference on artificial intelligence* (pp. 3094–3100). AAAI Press.

Maaten, L. v. d., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, *9*(Nov), 2579–2605.

Mallat, S. (1999). *A wavelet tour of signal processing*. Elsevier.

Middlehurst, M., Large, J., Flynn, M., Lines, J., Bostrom, A., & Bagnall, A. (2021). HIVE-COTE 2.0: A new meta ensemble for time series classification. *Machine Learning*, *110*(11), 3211–3243.

Narwariya, J., Malhotra, P., Vig, L., Shroff, G., & Vishnu, T. (2020). Meta-learning for few-shot time series classification. In *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD* (pp. 28–36).

Paparrizos, J., & Gravano, L. (2017). Fast and accurate time-series clustering. *ACM Transactions on Database Systems*, *42*(2), 1–49.

Pei, H., Ren, K., Yang, Y., Liu, C., Qin, T., & Li, D. (2021). Towards generating real-world time series data. In *2021 IEEE international conference on data mining* (pp. 469–478). IEEE.

Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.

Ramponi, G., Protopapas, P., Brambilla, M., & Janssen, R. (2018). T-CGAN: Conditional generative adversarial network for data augmentation in noisy time series with irregular sampling. ArXiv, arXiv:1811.08295.

Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2014). Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806.

Tan, C. W., Dempster, A., Bergmeir, C., & Webb, G. I. (2022). MultiRocket: Multiple pooling operators and transformations for fast and effective time series classification. *Data Mining and Knowledge Discovery*, *36*(5), 1623–1646.

Tang, W., Long, G., Liu, L., Zhou, T., Blumenstein, M., & Jiang, J. (2022). Omni-scale CNNs: A simple and effective kernel size configuration for time series classification. In *ICLR*.

Wang, H., Li, S., Wang, T., & Zheng, J. (2021). Hierarchical adaptive temporal-relational modeling for stock trend prediction. In *30th international joint conference on artificial intelligence*. International Joint Conferences on Artificial Intelligence.

Wiese, M., Knobloch, R., Korn, R., & Kretschmer, P. (2019). Quant GANs: Deep generation of financial time series. arXiv preprint arXiv:1907.06673.

Wilcoxon, F. (1992). Individual comparisons by ranking methods. In *Breakthroughs in statistics* (pp. 196–202). Springer.

Xu, T., Wenliang, L. K., Munn, M., & Acciaio, B. (2020). Cot-gan: Generating sequential data via causal optimal transport. *Advances in Neural Information Processing Systems*, *33*, 8798–8809.

Ye, S., Hu, X., & Xu, X. (2020). Tdcgan: Temporal dilated convolutional generative adversarial network for end-to-end speech enhancement. arXiv preprint arXiv:2008.07787.

Yeh, C.-C. M., Van Herle, H., & Keogh, E. (2016). Matrix profile III: The matrix profile allows visualization of salient subsequences in massive time series. In *2016 IEEE 16th international conference on data mining* (pp. 579–588). IEEE.

Yoon, J., Jarrett, D., & Van der Schaar, M. (2019). Time-series generative adversarial networks. *Advances in Neural Information Processing Systems*, *32*.

Yue, Z., Wang, Y., Duan, J., Yang, T., Huang, C., & Xu, B. (2021). Towards universal representation of time series. In *AAAI*.

Zhang, X., Yao, L., & Yuan, F. (2019). Adversarial variational embedding for robust semi-supervised learning. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 139–147).