Efficient Clustering of Multivariate Variable Length Time Series with Representation Learning

Anonymous Author(s)

ABSTRACT

Time series clustering is important for any domain where category labels are rarely or sparsely available and detecting patterns is critical. Recently, deep learning models have shown notable results in supervised tasks, by using representation learning. However, ensuring that such a representation is cluster-friendly and effectively captures the temporal and multi-dimensional dynamics of variable length time series, remains a challenge. We propose a fully unsupervised and efficient approach that simultaneously optimizes for both representation learning and clustering tasks. This is achieved by combining a stacked dilated causal CNN encoder-only architecture with a composed loss function with cosine similarity-based negative sampling and iterative training. Extensive experiments show that our approach outperforms existing methods in at least 70% of the cases, and is comparable otherwise. Further, we prove its efficiency and scalability on series with millions of datapoints and show that it is particularly suitable for real-life industrial applications.

CCS CONCEPTS

• **Computing methodologies** → **Cluster analysis**; **Neural networks**; **Learning latent representations**.

KEYWORDS

multivariate time series, variable length, clustering, neural networks, composed loss function

ACM Reference Format:

Anonymous Author(s). 2021. Efficient Clustering of Multivariate Variable Length Time Series with Representation Learning. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Temporal data is everywhere, ranging from ECG signals in healthcare, to sensor metrics collected from IoT devices or real-time stock indicators in the financial market. More often than not, such data is *multivariate* and a particular behavior is characterized by multiple dimensions, typically hundreds or even thousands.

Because of their volume and variety, real-life time series are *rarely or sparsely labeled*, which limits the spectrum of learning approaches that can be used to unsupervised or at best semi-supervised. In addition, these series typically have *variable lengths*. Common examples are medical electronic records captured for the duration of a patient's illness, or abnormal patterns in performance of large-scale data centers that are sustained over arbitrary periods of time (e.g., ranging from minutes to days or even weeks). In the latter case, depending on the complexity of the environment, some deviations from the normal behavior need to propagate through multiple layers until they are noticeable at the client side or by support staff (e.g., backup problems), as opposed to others which may immediately disrupt operation (e.g., network interruptions, increased latency and response time, overloaded CPU, etc.).

Considering the above constraints – *few or no labels*, *high dimensionality*, *variable length* of the time series, as well as *variety of behavior patterns* in the data –, in this work we investigate the topic of fully unsupervized clustering for multivariate variable length time series via representation learning.

Clustering is relevant for any domain where pattern detection is critical, such as diagnosis and prognosis in healthcare or anomaly detection and prediction in industry and IoT scenarios, just to name a few. Most existing methods for time series clustering are domain-dependent and require domain knowledge to manually engineer discriminative features from input data [20, 22]. However, such features are typically linear, while time series mostly exhibit non-linear dynamics [3, 15].

In recent years, deep learning models have been used for time series learning with notable results [10, 16, 17, 29]. In particular, some of the existing approaches have shown that representation learning can significantly improve the downstream task (i.e., typically classification).

Inspired by these findings, we propose to learn a non-linear temporal representation for clustering. Given the lack of labels in practice, effectively guiding the learning process of the deep learning model to generate cluster-friendly representations (i.e., compatible embeddings for similar time series and linearly separable ones otherwise), as well as capturing the dynamics and multi-scale characteristics of time series is a challenge. We introduce a novel fully unsupervised and efficient temporal clustering based on representation learning. The novelty of our approach is in the integration of the temporal representation learning and clustering tasks within a stacked exponentially dilated causal CNN encoder-only model via a composed loss function and iterative training. More specifically, the contributions of this work are as follows:

- a composed loss function that optimizes for both representations learning and clustering objectives, inspired by word2vec and cosine similarity-based negative sampling;
- a fast encoder-only CNN with dilated causal convolutions that can handle long and variable length time series;
- simultaneous optimization for learning fixed-length representations and performing the clustering task through iterative training.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

We evaluate our model against well-known traditional time series clustering approaches, as well as recent deep learning models, on various datasets from the UCR [6] (univariate time series) and UEA [1] (multivariate time series) benchmarks. Results show that we outperform all considered methods in at least 70% of the cases and achieve comparable performance otherwise, *without any dataset-specific parameter tuning*. To demonstrate the efficiency and scalability of our encoder-only model, we apply it on a real-life dataset that includes very long time series (i.e., over 2 million datapoints).

This paper is organized as follows. Section 2 outlines previous works on distance-based clustering, deep learning-based clustering and representation learning for time series. Section 3 describes our architecture, as well as the unsupervized training of the encoder. Section 5 reports evaluation results of the approach relative to selected state-of-the-art methods.

2 RELATED WORK

Distance-based clustering – Measuring the distance between different time series needs to take into consideration the temporal correlation between the data points in a time series. In the simplest case, when the time series have the same length and are sampled at the same frequency, general clustering methods based on Euclidean distance, such as k-means, can be used. However, when the sampling rates between comparing time series are different, elastic distance measures such as dynamic time warping (DTW) [24], edit distance on real sequence (EDR) [5] or edit distance with real penalty (ERP) [4], as well as cross-correlation measures such as the SBD metric used in k-shape [20] are preferred.

Deep learning-based clustering – Deep learning approaches have been proposed in recent years to deal in particular with high-dimensional data, from images to time series. The idea is to encode the input data into latent representations that are not only more compact, but can drive the clustering task towards better performance. The first notable attempt is DEC [33], which uses stacked denoising autoencoders to generate representative embeddings. However, spatial relationships in the reconstructed samples cannot be guaranteed. To alleviate this drawback, IDEC [11] defines a joint optimization objective that minimizes KL divergence simultaneously with the reconstruction loss. Similar to IDEC, DCC [28] and DCN [34] use the reconstruction loss as part of the optimization objective. In DCC, the authors assume the number of clusters is unknown in advance and propose a clustering algorithm that continuously performs the clustering task by optimizing an objective that needs no updating during the optimization phase. DCN uses a novel alternating stochastic gradient algorithm and a cluster structure-promoting regularization to achieve superior accuracy compared to IDEC. Differently, DEPICT [7] uses discrete reconfigurations of datapoints to centroids and in [35] the authors use agglomerative clustering.

Recently, DTC [17] specifically tackles time series clustering. The authors use an autoencoder for temporal dimensionality reduction coupled with a clustering layer for cluster assignment. The clustering layer can be customized with one of four similarity metrics. Based on dataset and application, the performance of using one or the other metrics can significantly vary and ideally the best metric to apply should already be known (i.e., this requires expensive parameter tuning). In addition, computing the cluster assignments is based on KL divergence and the same target distribution as defined in DEC. However, this distribution may not capture all the dynamics of the input time series (i.e., reflected in the comparative performance results in Section 4).

Representation learning – Some recent works tackle unsupervised representation learning for time series. [12] learn representations on evenly sized subdivisions of time series by learning to discriminate between these subdivisions. [14] propose a method designed so that the distances between learned representations mimic a standard distance (DTW) between time series. [18] design an encoder as a recurrent neural network, jointly trained with a decoder as a seq2seq model to reconstruct the input time series from its learned representation. [32] compute feature embeddings generated in the approximation of a carefully designed and efficient kernel. However, the use of recurrent networks or of DTW (with its quadratic complexity) hinder these methods' ability to scale well. To alleviate these drawbacks, [10] propose an encoder-based architecture based on causal convolutions inspired by [29] to ensure faster training and inference. This model used in conjunction with a triplet loss function is shown to perform very well for classification tasks on various time series datasets, but has not been applied for clustering.

Last but not least, a representation learning approach for time series clustering, called DTCR, has been proposed in [16]. The temporal reconstruction and k-means objective are integrated into a seq2seq model. The goal is very similar to ours, that is to learn representations that lead to improved cluster structures. To enhance the ability of their encoder, the authors include a fake-sample generation strategy and auxiliary classification task. Minimizing the classification error becomes part of the composed loss function, in addition to the reconstruction error of the decoder. We, on the other hand, opt for an encoder-only architecture, which is more computationally efficient, and incorporate the clustering loss and the error of learning representations in the model's objective.

3 APPROACH

Let $D = \{y_1, ..., y_n\}$ be a set of N time series. Each series y_i contains a number T of ordered values, denoted as $y_i = (y_{i,1}, ..., y_{i,T})$, where T can vary from series to series. Define the non-linear mapping f: $y_i \rightarrow h_i$, which encodes the input time series into an m-dimensional latent representation, such that $h_i = f(y_i)$, $h_i \in \mathrm{R}^m$. The goal is to learn representations that facilitate a clustering task where all time series need to be assigned within K clusters.

Our approach incorporates an encoder-only architecture (Section 3.1), coupled with a composed loss function (Section 3.2) and iterative training (Section 3.3).

3.1 Architecture

The choice of architecture is motivated by a few constraints: 1) it should be able to extract the dynamics and multi-scale characteristics of time series; 2) must be computationally efficient, especially for clustering tasks of very long time series; 3) must allow for variable length inputs. As a result, we use a convolutional encoder-only network with exponentially dilated causal convolutions.

Recurrent neural networks have been preferred for time series, since they are specifically designed for sequence-modeling tasks.

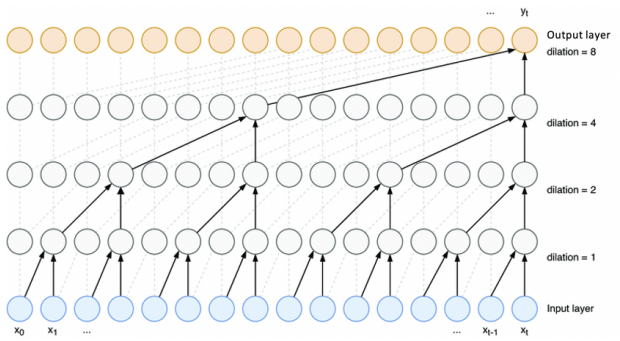


Figure 1: Stacked causal dilated convolutions.

However, they suffer from short memory, convergence and vanishing/exploding gradients issues. Convolutional networks, on the other hand, train and converge faster and allow efficient parallelization on modern hardware such as GPUs. Even more, exponentially dilated convolutions can better capture long-range dependencies at constant depth by exponentially increasing the receptive field of the network [2, 29, 36]. Our proposed architecture is based on stacked dilated causal convolutions (Fig. 1). By using causal convolutions, we ensure that any element of the output sequence is computed using only values up to its index in the input sequence.

Inspired by [2, 10], each layer of the network is a combination of causal convolutions, weight normalizations [25], leaky ReLUs and residual connections, as shown in Fig. 2. Each layer has a dilation parameter of the type 2^i for layer i. The output of the network is then passed through a global max pooling layer. This squeezes the temporal dimension and aggregates all temporal information in a fixed-sized vector. A linear transformation of the vector represents the learnt representation and it is passed to a clustering layer that performs the clustering task. Any clustering algorithm can be used – with no specific considerations for temporal data, different sampling frequencies or lengths –, since the learnt representations are fixed length and comparable in the Euclidean space.

The goal of the encoder is to ensure that similar time series are represented by similar learnt representations, without any supervision to learn such similarity. Most existing approaches treat representation learning and clustering as separate and consecutive tasks. Such models optimize to generate representations that minimize the reconstruction loss of the decoder. In addition to the larger computational costs incurred for both training and inference when using a decoder (which we address by using an encoder-only architecture), even more important is the fact that the clustering objective is not actively considered in the representation learning process.

To alleviate the latter limitation, we aim to perform a joint and simultaneous optimization for both representation learning and clustering. This is achieved by using a composed loss function and iterative training, as detailed next.

3.2 Composed loss function

Composed losses have been used recently for representation learning [19, 26, 31], but not in the context of temporal data and not in a fully unsupervized setting until [10]. Such losses require pairs of similar inputs, which in turn typically need annotated data. In [10], concepts from word2vec [19] (i.e., in particular the CBOW model) are used to select such pairs.

Adapted to time series, consider a subseries y^{ref} of a given series y_i. Then, the composed loss needs to ensure that the representation of y^{ref} is close to any of its subseries y^{pos} and distant from a subseries y^{neg} chosen from another series y_j. To increase the stability of the training step and ensure a higher degree of separation in the learned representations of series belonging to different clusters, we choose multiple $(y_p^{neg})_{p\in[1,P]}$ samples.

We therefore write the loss function corresponding to learning representations as in [10]:

$$\begin{aligned} \mathsf{L}_{\mathsf{repr}}^{\mathsf{y_{ref}}} = &-log(\sigma(f(y^{ref},\theta)^{\mathsf{T}} f(y^{pos},\theta))) \\ &- \sum_{p=1}^{P} log(\sigma(-f(y^{ref},\theta)^{\mathsf{T}} f(y_p^{neg},\theta))) \end{aligned} \tag{1}$$

where θ are the parameters of the encoder and σ is the sigmoid function. The goal is to push the learnt representations to distinguish between y^{ref} and y^{neg}, while at the same time to assimilate y^{ref} and y^{pos}.

The length of y^{neg} samples is chosen to match that of y^{pos}, in order to speed the training process. We highlight the process of choosing y^{ref}, y^{pos} and $\{y^{neg}\}_p$ in Algorithm 1.

Algorithm 1 Choosing y^{ref}, y^{pos} and $\{y^{neg}\}p$ within an epoch

for i ∈ [1,N] with s_i = size(y_i) **do**
  s_{pos} = len(y^{pos}) ∈ [1,s_i]
  s_{ref} = len(y^{ref}) ∈ [s_{pos},s_i]
  Choose y^{ref} randomly from y_i with s_{ref}
  Choose y^{pos} randomly from y_i with s_{pos}
  Choose s_p^{neg} = len(y_p^{neg}) ∈ [1,$len(y_p)$]
  Choose y_p^{neg} from y_p with s_p^{neg} and $cos_{sim}(y_p^{neg}, y^{ref}) < th$

While y^{pos} is chosen randomly from the same series as y^{ref}, to select $\{y^{neg}\}_p$, we use cosine similarity. More specifically, if the similarity between y_p^{neg} and y^{ref} is lower than th, y_p^{neg} is a desirable negative sample to be used in the representation learning process (i.e., the closer is the cosine similarity to 0, the less similar the compared series are). We set $th = 0.4$. The selection process of $\{y^{neg}\}_p$ finishes when we have found P subseries fulfilling the similarity constraint. In the exceptional case where this is not possible (i.e. series in the dataset are too similar), we reduce th with exponential decay and repeat the process.

So far, the loss function enables the encoder to take as input time series of variable lengths. By training the network on a range of input lengths ranging from the shortest to the longest time series in the set, it becomes able to output meaningful representations regardless of the input length.

The second part of the composed loss function minimizes the clustering loss. If k-means is used, then we adapt its cost function, for each y^{ref} to the following formulation:

$$\mathsf{L}_{\mathsf{cl}}^{\mathsf{y^{ref}}} = ||f(y^{ref}) - Ms_{ref}||_2^2 \tag{2}$$

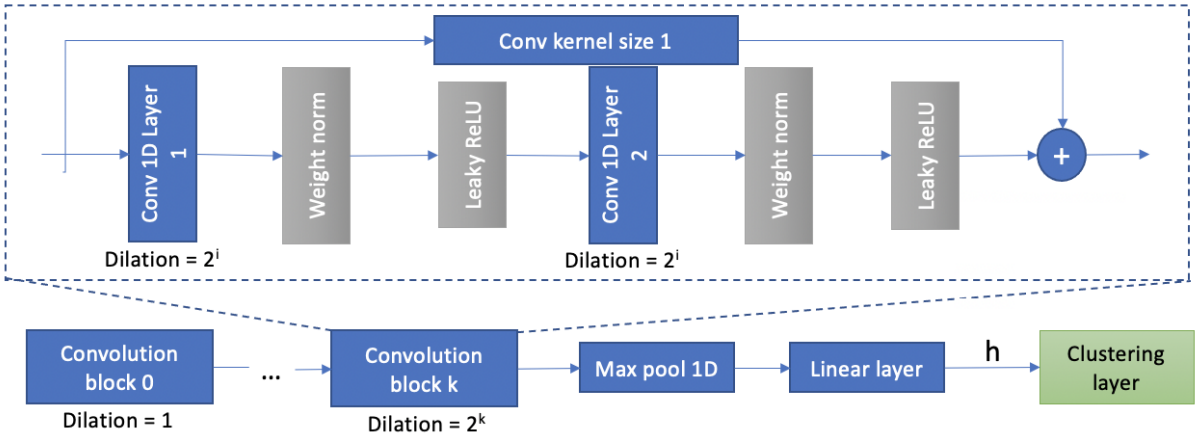


Figure 2: Architecture of our approach.

where s_{ref} is the assignment vector of time series y^{ref} which has only one non-zero element (i.e., since a series can only belong to one cluster) and M is the list of centroids with the m_k element denoting the centroid of the kth cluster. Of course, other clustering criteria, such as soft k-means or DBSCAN [9] are also viable options, but require adapting the clustering loss function.

Finally, we write the composed loss function as a combination of $\mathsf{L}_{\mathtt{repr}}$ and $\mathsf{L}_{\mathtt{cl}}$:

$$\underset{\theta,M,s_{ref}}{\mathsf{L}} = min \sum_{y^{ref}}^{N} (\mathsf{L}_{\mathtt{repr}}^{y^{ref}} + \lambda \mathsf{L}_{\mathtt{cl}}^{y^{ref}}) \tag{3}$$

where λ is a regularization parameter which balances the representation error versus finding k-means-friendly latent representations.

3.3 Iterative training

Solving Eq. 3 is challenging and stochastic gradient descent (SGD) cannot be directly applied to jointly optimize θ, M and s_{ref}, because s_{ref} is constrained to a discrete set. We approach this problem by alternating optimization and SGD, namely by optimizing with respect to one of the variables, while keeping the other(s) fixed.

3.3.1 Updating encoder parameters. For a fixed set (M, s_{ref}), the problem becomes training the encoder, with an additional penalty term on the clustering performance. We update the network parameters with respect to the incoming time series y^{ref}:

$$\underset{\theta}{min}(\mathsf{L}^{y^{ref}}) = \mathsf{L}_{\mathtt{repr}}^{y^{ref}} + \lambda \mathsf{L}_{\mathtt{cl}}^{y^{ref}} \tag{4}$$

The gradient is computed as $\nabla_\theta \mathsf{L}^{y^{ref}} = \frac{\partial \mathsf{L}_{repr}^{y^{ref}}}{\partial \theta} + \lambda \frac{\partial f(y^{ref})}{\partial \theta}(f(y^{ref}) - Ms_{ref})$. The encoder parameters are then updated by

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathsf{L}^{y^{ref}} \tag{5}$$

where $\alpha > 0$ is a decreasing learning rate.

3.3.2 Updating clustering parameters. Next, while keeping the network parameters fixed, we alternatively update the centroids M and the assignment vector of the current time series s_{ref}. Updating s_{ref} can be done in an online fashion as follows:

$$s_{j,ref} \leftarrow \begin{cases} 1, \text{if } j = \underset{k=1,\ldots,K}{argmin} ||f(y^{ref}) - m_k||_2 \\ 0, \text{otherwise} \end{cases} \tag{6}$$

When θ and s_{ref} are fixed, updating M can be done in multiple ways. The simplest approach is to simply average the current assigned samples. However, in cases where already considered samples are not representative enough to model the global cluster structure, this solution can be problematic. Rather, we adapt the idea proposed in [27] to adaptively change the learning rate of updating the centroids $m_1, ..., m_K$. In a nutshell, assume that the clusters are roughly balanced in terms of the number of time series samples they contain. After updating M for a number of samples, we update the centroids of the clusters with many assigned members more gracefully while updating others more aggressively, to strive for balance. Let c_k^{ref} be the number of times a sample has been assigned to cluster k before handling y^{ref}. We update m_k by:

$$m_k \leftarrow m_k - (\frac{1}{c_k^{ref}})(m_k - f(y^{ref}))s_{k,ref} \tag{7}$$

where the gradient step size $\frac{1}{c_k^{ref}}$ controls the learning rate.

We summarize the iterative training algorithm in Algorithm 2. Note that an epoch corresponds to a pass of all data samples through the network.

Algorithm 2 Iterative training

```
for t=1:T do                          ▷ T = number of epochs
    Update parameters θ by (5)
    Update assignments s^ref by (6)
    Update centroids m_k by (7)
```

4 EVALUATION

We evaluate the performance of our clustering approach on a variety of univariate and multivariate time series data sets. We train the model with the same set of parameters independent of the data set, since we perform no hyperparameter optimization on the encoder, unlike other methods like [18]. As optimizer we use Adam [13] with a learning rate $\alpha = 0.001$ and decay rates $\beta = (0.9, 0.999)$. To train the encoder, we use a batch size of 10. The number of optimization steps depends on how many negative samples, P, we use in the composed loss function. As recommended in [10], lower values for P can be used for univariate series sets (e.g., $P = \{2,5,10\}$) and higher values for multivariate series (e.g., $P = \{5,10,20\}$). Therefore, when $P = 10$, we typically perform 3000 optimization steps (i.e., for a dataset of size 1000, this implies 30 epochs).

In terms of architecture, the number of channels used in the intermediary layers of the causal encoder is 40 and the total number of layers, namely the actual depth of the network, is 10. All convolutions have a kernel size of 3. The negative slope of the leaky ReLU activation function is set to 0.01. The number of output channels of the causal network before max pooling is 320 and the final dimensions of the learned representations is 160. Finally, we set the weight of the clustering loss to 1, which implies that we treat the clustering objective and the goal of learning representations as equally important. We detail the influence of this weight in Section 4.6.

The model is implemented in Pytorch version 0.4.1 [21]. We train the causal encoder on one Nvidia GeForce GTX 1080 Ti GPU.

4.1 Evaluation metric

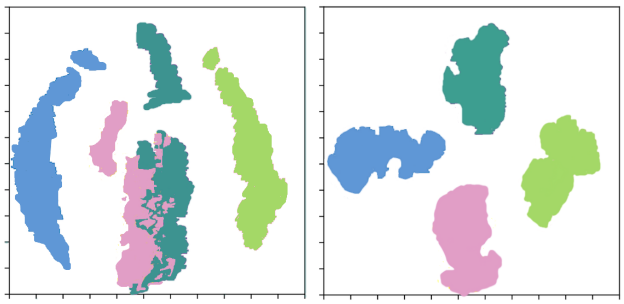
To evaluate clustering performance, the metric considered should find the best matching between ground truth labels (i.e., as provided in the datasets) and cluster assignments, as outputted by the model. To this end, we use Rand Index [23] (RI). RI is defined as follows:

$$RI = \frac{TP + TN}{N(N-1) + 2} \in [0, 1] \tag{8}$$

where TP (True Positive) is the number of pairs of time series that are correctly put in the same cluster, TN (True Negative) is the number of pairs that are correctly put in different clusters and N is the size of the data set. Values close to 1 indicate high quality clustering [37], therefore are desired.

4.2 Representative approaches

We compare our approach with 6 representative time series clustering methods: 1) **DTW** [24] – classic algorithm that measures similarity between two temporal sequences, which may vary in sampling frequency; 2) **k-shape** [20] – adopts a scalable iterative refinement procedure to compare time series based on their shape by using a normalized version of the cross-correlation measure; 3) **DEC** [33] – learns a mapping from the data space to a lower-dimensional feature space in which it optimizes a clustering objective in an iterative fashion; 4) **IDEC** [11] – to maintain the local structure of the underlying data, the reconstruction error of the autoencoder and the clustering loss are jointly used to optimize the cluster labels assignment; 5) **DTC** [17] – takes the KL divergence between the predicted and the target data distributions as guidance to learn non-linear features in a deep framework composed of convolutions and bidirectional LSTM layers; 6) **DTCR** [16] – integrates the temporal reconstruction and k-means objective into a seq2seq model to learn cluster-specific temporal representations and improve cluster structure.



(a) `DTCR` **(b) `Ours`**

Figure 3: Latent representations after training for DTCR (RI = 0.7) and our approach (RI = 1) on the `TwoPatterns` UCR dataset. Visualizations are obtained with t-SNE (perplexity 30).

4.3 Results for univariate time series

First, we show accuracy scores on 36 datasets from the new UCR archive [6] in Table 1. While the benchmark contains 128 datasets, the most representative approach we compare against, DTCR, only evaluates on 36 of them. For these, we include results for k-shape, DEC, IDEC, DTC and DTCR as reported in [16]. In the case of DTW, we run the algorithm ourselves across all datasets.

Our approach achieves the best average RI (0.86), followed by DTCR (0.77) and DTW (0.76). In 72% of the cases (i.e., 26 out of 36), we achieve the best performance in comparison to all other methods, whereas in the remaining 28% of datasets, DTCR typically performs best, but mostly by slim margins between 2% and 6%). Fig. 3 shows the latent representations learned by our model and DTCR on the `TwoPatterns` dataset, which contains 4 clusters. As seen, our approach obtains perfectly separated embeddings for the 4 groups, while DTCR produces partially overlapping representations for 2 of the clusters. These observations are reflected in the RI scores, namely RI = 1 compared to RI = 0.7.

Additionally, we make the following observations. First, DEC and IDEC achieve worse results in general than all other methods, with the exception of k-means. This is to be expected, since none of these approaches are designed to deal specifically with temporal data. Second, both k-shape and DTC achieve significantly lower performance than DTW, DTCR and our approach (i.e., 10%, 11% and 20% loss on RI, respectively). Even more so, they are comparable with DEC and IDEC. This is surprising, since both methods are built for time series.

With respect to DTC, the authors use the same clustering loss function based on KL divergence as proposed in DEC. In particular the definition of the target distribution does not necessarily match well with temporal data, and therefore it will not ensure that the learned embeddings capture enough of the dynamics and characteristics of this type of data modality. In addition, the approach proposes four different similarity metrics that one can use with wildly varying accuracy (Table 1, Section 4.5 in [17] for UCR results). None of these metrics consistently perform better than the rest across all datasets, which is why we report the average performance of all four

Dataset	(Length, # clusters)	k-means	DTW	k-shape	DEC	IDEC	DTC	DTCR	Ours
ArrowHead	(252,3)	0.69	0.7	0.73	0.58	0.62	0.67	0.69	**0.77**
Beef	(471,5)	0.67	0.63	0.54	0.6	0.63	0.63	**0.8**	0.77
BeetleFly	(513,2)	0.48	0.7	0.61	0.49	0.61	0.7	0.9	**0.93**
BirdChicken	(513,2)	0.49	0.75	0.66	0.47	0.48	0.73	0.81	**0.85**
Car	(578,4)	0.63	0.74	0.7	0.69	0.69	0.67	0.75	**0.83**
ChlorineConcentration	(167,3)	0.52	0.65	0.41	0.53	0.54	0.54	0.54	**0.75**
Coffee	(287,2)	0.75	**1**	**1**	0.49	0.58	0.48	0.93	**1**
DiatomsizeReduction	(346,4)	0.96	0.97	1	0.93	0.74	0.88	0.97	**0.98**
Dist.Pha.Out.Agegroup	(81,3)	0.62	**0.78**	0.6	**0.78**	**0.78**	**0.78**	**0.78**	0.73
DistalPha.Out.Corr.	(81,2)	0.53	0.72	0.53	0.5	0.53	0.5	0.61	**0.77**
ECG200	(97,2)	0.63	0.77	0.7	0.64	0.62	0.6	0.67	**0.93**
ECGFiveDays	(137,2)	0.48	0.93	0.5	0.51	0.51	0.5	**0.96**	0.93
GunPoint	(151,2)	0.5	0.91	0.63	0.5	0.5	0.54	0.64	**0.97**
Ham	(432,2)	0.5	0.5	0.53	0.6	0.5	0.56	0.54	**0.72**
Herring	(513,2)	0.5	0.53	0.5	0.51	0.51	0.5	**0.58**	**0.58**
Lighting2	(638,2)	0.5	**0.87**	0.65	0.53	0.55	0.58	0.6	**0.87**
Meat	(449,3)	0.66	0.93	0.66	0.65	0.62	0.32	**0.98**	0.95
Mid.Pha.Out.Agegroup	(81,3)	0.54	0.5	0.51	0.71	0.68	0.58	**0.8**	0.66
MiddlePha.Out.Corr.	(81,2)	0.5	0.7	0.51	0.54	0.54	0.53	0.56	**0.83**
MiddlePhalanxTW	(81,6)	0.1	0.5	0.62	**0.86**	**0.86**	0.71	**0.86**	0.62
MoteStrain	(85,2)	0.49	0.84	0.61	0.74	0.73	0.75	0.77	**0.85**
OSULeaf	(428,6)	0.56	0.59	0.55	0.75	0.76	0.73	**0.78**	0.76
Plane	(145,7)	0.91	0.99	0.99	0.94	0.94	0.9	0.96	**0.99**
Prox.Pha.Out.Agegroup	(81,3)	0.53	0.8	0.56	0.43	0.81	0.74	0.81	**0.84**
ProximalPhalanxTW	(81,6)	0.48	0.78	0.52	0.82	0.9	0.84	**0.9**	0.85
SonyAIBORobotSurface	(71,2)	0.77	0.74	0.81	0.57	0.69	0.8	0.88	**0.9**
SonyAIBORobotSurfaceII	(66,2)	0.87	0.83	0.56	0.65	0.66	0.8	0.84	**0.89**
Symbols	(399,6)	0.88	0.95	0.84	0.88	0.89	0.91	0.92	**0.96**
SwedishLeaf	(129,15)	0.5	0.79	0.53	0.88	0.89	0.89	**0.92**	0.91
ToeSegmentation1	(278,2)	0.49	0.77	0.61	0.5	0.5	0.51	0.57	**0.94**
ToeSegmentation2	(344,2)	0.53	0.84	0.53	0.5	0.5	0.53	0.83	**0.89**
TwoPatterns	(129,4)	0.85	0.98	0.8	0.63	0.63	0.63	0.7	**1**
TwoLeadECG	(83,2)	0.55	0.9	0.82	0.5	0.5	0.51	0.71	**0.99**
Wafer	(153,2)	0.49	0.98	0.49	0.57	0.56	0.53	0.73	**0.99**
Wine	(235,2)	0.5	0.57	0.5	0.49	0.52	0.49	0.63	**0.82**
WordsSynonyms	(271,25)	0.88	0.65	0.78	0.89	0.89	0.89	**0.9**	0.84
Avg. RI	-	0.6	0.76	0.64	0.64	0.65	0.66	0.77	**0.86**

Table 1: Rand Index (RI) results on 36 UCR datasets, compared against representative approaches.

models (i.e., as done in [16]). Of course with parameter optimization and ensuring that the best similarity metric per dataset is always used, accuracy can be boosted. However, this defeats the purpose of having a single model that consistently performs well across a variety of benchmarks, without parameter tuning.

With respect to k-shape, depending on how the original time series needs to be shifted and scaled to align properly, the algorithm can encounter convergence issues. While the authors suggest that a small number of iterations (e.g., 100) are typically enough to obtain a stable clustering, for some datasets it might require a significantly larger number of iterations or might not converge at all, hence the lower RI.

Influence of P – The number of negative samples randomly chosen can vary significantly. In the simplest case, P can be set to 1. However, as noted in [10], $P > 1$ will yield better performance. Depending on the length of the time series, smaller values of P (e.g., $P \in [2,5]$) can be chosen for shorter series and higher values ((e.g., $P \in [5,10]$) for longer series. The datasets in UCR vary wildly in length – with $T_{\texttt{SonyAIBORobotSurface}} = 71$ and $T_{\texttt{Lighting2}} = 638$ –, and using different values of P can impact RI. However, we choose $P = 5$ across all datasets, since we do not perform any parameter optimization of the model prior to training. Of course, one could generate latent representations for each P value in a discrete set and use a concatenation of these embeddings for the clustering task. However, this is non-trivial, since our approach optimizes for representation learning and clustering in an iterative manner. At the same time, even without such parameter tuning, we outperform other methods in a majority of cases.

Dataset	(Length, # cl.)	DTW_D	DTCR	Ours
Cricket	(1197,12)	**1**	0.94	0.97
DuckDuckGeese	(270,5)	0.6	0.55	**0.74**
EigenWorms	(17984,5)	0.62	0.7	**0.86**
Epilepsy	(206,4)	0.96	0.93	**0.97**
EthanolConcentr.	(1751,4)	**0.36**	0.22	0.34
Handwriting	(152,26)	0.29	0.4	**0.54**
Heartbeat	(405,2)	0.72	0.67	**0.79**
SelfReg.SCP2	(1152,2)	**0.56**	0.47	**0.56**
Phoneme	(217,39)	0.15	**0.26**	0.25
StandWalkJump	(2500,3)	0.2	0.4	**0.47**
Avg. RI	-	0.55	0.55	**0.65**

Table 2: RI results on 10 UEA datasets with fixed length, when compared with DTW_D and DTCR.

Dataset	(Length, # cl.)	DTW_D	Ours
CharacterTraj.	(109-205,20)	**0.99**	**0.99**
JapaneseVowels	(7-29,9)	0.95	**0.99**
SpokenArabicDigits	(4-93,10)	0.94	**0.95**
Avg. RI	-	0.96	**0.98**

Table 3: RI results on the 3 UEA datasets with variable length, when compared with DTW_D.

4.4 Results for multivariate time series

Since UCR contains only univariate time series and our approach is designed specifically for multivariate inputs, we evaluate it on UEA [1] as well. This benchmark contains sets with both fixed and variable length series. Some representative approaches, such as k-means, DEC or IDEC, are not able to handle multivariate inputs at all. Others, like DTC and DTCR, while able to handle higher input dimensionality, cannot be applied for variable length inputs (due to limitations in their respective architectures). As discussed in Section 4.3, in an ideal scenario the choice of P depends on the length of the time series per dataset. However, as before, we choose a fixed value (i.e., $P = 10$, since UEA series are typically longer than time series in UCR) across all datasets and report results next.

Fixed length – Since DTCR and DTW performed best on UCR, we choose these approaches to evaluate against. For DTW, we use dimension-dependent DTW, DTW_D, to handle multivariate inputs. Results for 10 UEA datasets are shown in Table 2. Our approach outperforms both DTCR and DTW_D as reflected in the average RI and the fact that it achieves best absolute results on 70% of the datasets. At the same time, we note that overall DTCR and DTW_D perform similarly (i.e., same as for UCR), even though there may be significant differences between them on individual datasets (e.g., on `EthanolConcentration` DTW_D is superior to DTCR, and vice versa on `StandWalkJump` or `Handwriting`).

Variable length – Results for our approach and DTW_D on the 3 datasets in UEA with variable length series are shown in Table 3. As expected, we again outperform DTW_D, but by a significantly smaller margin (2%). This is due to the fact that the clusters in these particular datasets are visually separable, making the clustering task rather easy for both methods.

Due to the convolutions and max pooling layers used in our encoder, the architecture can naturally ingest variable length inputs. However, as noted earlier, methods like DTCR have architecture limitations that prohibits them to handle such constraints. In order to enable their applicability on such data, one would either have to: 1) modify the architecture correspondingly (e.g., convolution filters slide over the input series, independent of the length), or 2) transform the input series to the same fixed length, either by padding to the longest length or truncating to match the shortest length. Neither of these alternatives are desirable. In particular, padding or truncating time series can be non-trivial, as these should ideally preserve the underlying data distribution. Depending on the methodologies applied and how significant is the gap between the shortest and longest series, such preservation may not always be feasible.

4.5 Results on long time series

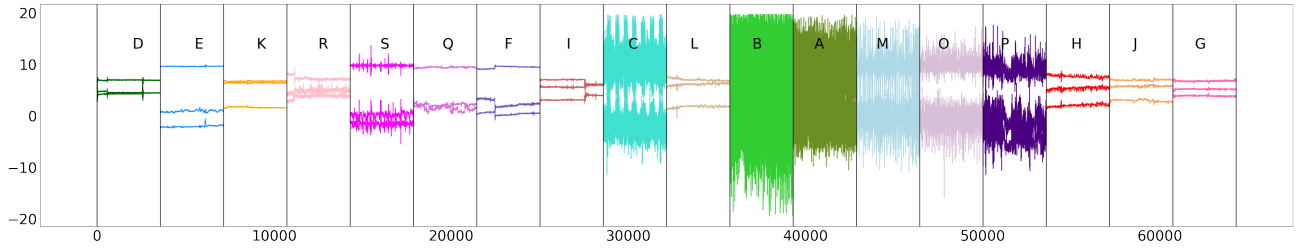
So far, we have shown that our approach is effective in extracting the dynamic and multi-scale characteristics of multivariate time series and in ingesting variable length inputs, while outperforming representative state-of-the-art methods. An equally important objective is to be computationally efficient. This is particularly critical for industrial applications where time series may contain even millions of datapoints and is the main reason behind our choice of an encoder-only architecture with exponentially dilated causal convolutions.

We complement our evaluation on UCR and UEA, whose datasets mostly contain short time series, with a scalability test on long series. The WISDM Smartphone and Smartwatch Activity and Biometrics dataset [30] from the UCI Machine Learning Repository [8] consists of 3-dimensional accelerometer sensors collected from 51 users' smartphones or smartwatches while performing 18 different activities (A-S in Fig. 4a). Each activity lasts 3 minutes and sensors are collected $\approx$ 20 times per second. These activities represent the clusters we aim to correctly identify. We concatenate the data of all users collected from their phones and obtain a 3-dimensional time series of more than 4.8 million datapoints. We split the series into train (data of first 10 users, approximately 960k measurements) and test (remaining measurements). When training the model, for each time step of the series, we compute the representations of the last window corresponding to 1 minute ($\approx$ 1200 measurements). The model is trained in around 6-8 hours, which shows that our approach is scalable enough to deal with very long series.

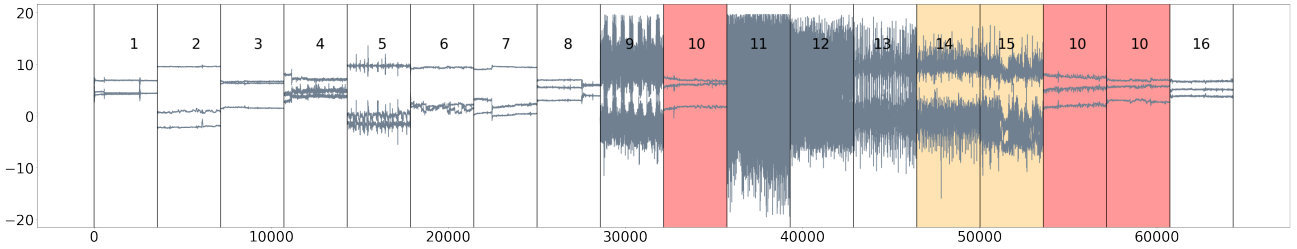
In Fig. 4b, we highlight the assigned clusters as detected by the encoder, relative to a user's activities (Fig. 4a). The model generally associates correctly different clusters with different activities, with two exceptions. In the case of activities L, H and J (i.e., eating sandwich/soup/pasta), the movements performed by the user are not distinguishable enough for the model to generate 3 different clusters. Finally, for activities O and P (i.e., playing catch and basketball), there is similarity in the hand movements a user would do, which results in 37% of the cases the model assigning the same cluster to both activities.

4.6 Influence of clustering loss

Finally, we investigate the influence of the clustering loss in the composed objective. In a first stage, we show the effectiveness of our



(a) Accelerometer sensors captured across 18 activities



(b) Detected clusters (with cluster IDs ∈ {1, ..., 16})

Figure 4: Detected clusters of human activity based on accelerometer data collected from users' smartphones. Activities are: A=Walking, B=Jogging, C=Stairs, D=Sitting, E=Standing, F=Typing, G=Brushing teeth, H=Eating soup, I=Eating chips, J=Eating pasta, K=Drinking, L=Eating sandwich, M=Dribbling, O=Playing catch, P=Basketball, Q=Writing, R=Clapping, S=Folding clothes.

loss function, whereby both L_{repr} and L_{cl} are minimized in an alternating manner. Thus, we compare our model on the UEA datasets and its ablation models: 1) model without L_{cl} (i.e., optimization focuses only on representation learning); and 2) model without L_{repr} (i.e., optimization focuses solely on the clustering task). We expect that optimizing for both representation learning and clustering tasks yields the best results. Results are reported in Table 4.

ID	Dataset	w/o L_{cl}	w/o L_{repr}	Both
A	Cricket	0.9	0.84	**0.97**
B	DuckDuckGeese	0.69	0.62	**0.74**
C	EigenWorms	0.78	0.77	**0.86**
D	Epilepsy	0.88	0.85	**0.97**
E	EthanolConcentr.	0.29	0.25	**0.34**
F	Handwriting	0.48	0.47	**0.54**
G	Heartbeat	0.65	0.6	**0.79**
H	SelfReg.SCP2	0.47	0.4	**0.56**
I	Phoneme	0.19	0.15	**0.25**
J	StandWalkJump	0.4	0.37	**0.47**
K	CharacterTraj.	0.96	0.94	**0.99**
L	JapaneseVowels	0.94	0.89	**0.99**
M	SpokenArabicDigits	0.91	0.86	**0.95**
	Avg. RI	0.66	0.61	**0.73**

Table 4: RI results on 13 UEA datasets when both losses are used in the objective function, compared to ablation models.

As seen, indeed our full model outperforms both ablation versions, indicating that the composed loss function and iterative training are effective. The average RI is 0.73 compared to 0.66 and 0.61, respectively. Compared to the model without L_{cl}, we obtain between 3% and 14% higher accuracy. The gap is even more significant relative to the model without L_{repr} (i.e., 5% - 19%). These results indicate that a model focused only on the clustering task will perform worse than a model that exclusively trains for representation learning. This is in fact not surprising. The quality of the generated representations – namely, their separability – is critical to ensure high clustering performance. In fact, one could use these representations as input to a subsequent clustering algorithm (i.e., clustering is performed after the encoder training) and obtain similar results as shown in column $w/o L_{cl}$ in Table 4.

In a second stage, we study the effect of λ in the composed loss function (Eq. 3). This represents the weight assigned to L_{cl} relative to L_{repr}. By default and in all previous experiments, we set λ to 1, meaning both L_{cl} and L_{repr} are equally important. In Fig. 5, we vary λ to $\{0.5, 1, 2, 4\}$ and show RI for all 13 UEA datasets (labeled A to M). For two datasets – `Phoneme` and `CharacterTrajectories` –, best performance is obtained when either $\lambda = 0.5$ or $\lambda = 1$. In all other cases, $\lambda = 1$ seems to be best option. This implies that our approach achieves best performance when the loss function is balanced. When $\lambda > 1$ and therefore the clustering loss weights significantly more, RI consistently drops by even 11-13%. This is in agreement with the results of our first ablation study, which found that a model focused solely on the clustering task will perform even worse than a model exclusively trained for representation learning.

5 CONCLUSIONS

In this paper, we proposed to learn non-linear temporal representations for clustering of multivariate, variable length time series, in a fully unsupervized setting. The novelty of our approach is in the integration (via an efficient composed loss function) and simultaneous optimization (through iterative training) of the representation learning and clustering tasks within an encoder formed by dilated

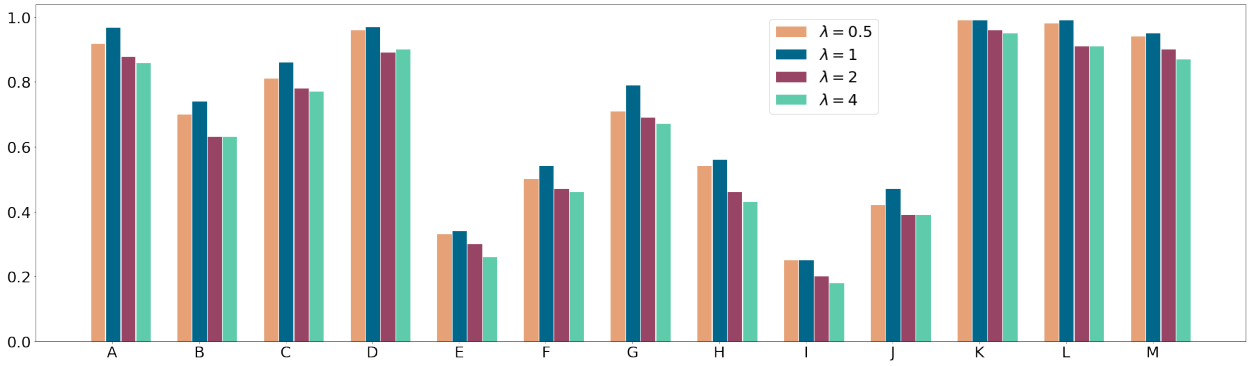


Figure 5: Effect of λ in the composed loss function to RI, for all 13 UEA datasets, when $\lambda \in \{0.5, 1, 2, 4\}$. Datasets are labeled A-M, based on their assigned IDs in Table 4.

convolutions. Extensive experiments showed the model's effectiveness when compared to state-of-the-art classic and deep-learning clustering methods. Furthermore, we proved the efficiency and scalability of our architecture on time series with millions of datapoints, making it particularly suitable for industrial applications.

REFERENCES

[1] Anthony Bagnall, Hoang Anh Dau, Jason Lines, Michael Flynn, James Large, Aaron Bostrom, Paul Southam, and Eamonn Keogh. 2018. The UEA multivariate time series classification archive. (2018).

[2] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. 2018. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271* (2018).

[3] Zongwu Cai, Jianqing Fan, , and Qiwei Yao. 2000. Functional-coefficient regression models for nonlinear time series. *J. Amer. Statist. Assoc.* 95(451) (2000), 1398–1409.

[4] Lei Chen and Raymond Ng. 2004. On the marriage of lp-norms and edit distance. *Proceedings of the Thirtieth International Conference on Very Large Databases* (2004), 792–803.

[5] Lei Chen, Tamer Ozsu, and Vincent Oria. 2005. Robust and fast similarity search for moving object trajectories. *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data* (2005), 491–502.

[6] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. 2018. The UCR Time Series Archive. (2018).

[7] Kamran Ghasedi Dizaji, Amirhossein Herandi, Cheng Deng, Weidong Cai, and Heng Huang. 2017. Deep Clustering via Joint Convolutional Autonecoder Embedding and Relative Entropy Minimization. (2017).

[8] Dheeru Dua and Casey Graff. 2017. UCI machine learning repository. (2017).

[9] Martin Ester, Hans-Peter Kriegel, Jiirg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *International Conference on Knowledge Discovery and Data Mining* (1996).

[10] Jean-Yves Franceschi, Aymeric Dieuleveut, and Martin Jaggi. 2019. Unsupervised Scalable Representation Learning for Multivariate Time Series. *Advances in Neural Information Processing Systems* 32 (2019), 4650–4661.

[11] Xifeng Guo, Long Gao, Xinwang Liu, and Jianping Yin. 2017. Improved Deep Embedded Clustering with Local Structure Preservation. *IJCAI* (2017).

[12] Aapo Hyvarinen and Hiroshi Morioka. 2016. Unsupervised feature extraction by time-contrastive learning and nonlinear ICA. *Advances in Neural Information Processing Systems* (2016), 3765–3773.

[13] Diedirik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. (2015).

[14] Qi Lei, Jinfeng Yi, Roman Vaculin, Lingfei Wu, and Inderjit S. Dhillon. 2017. Similarity preserving representation learning for time series analysis. *arXiv preprint arXiv:1702.03584* (2017).

[15] Qianli Ma, Sen Li, Lifeng Shen, Jiabing Wang, Jia Wei, Zhiwen Yu, and Garrison W. Cottrell. 2019. End-to-end incomplete time-series modeling from linear memory of latent variables. *IEEE Transactions on Cybernetics* (2019).

[16] Qianli Ma, Jiawei Zheng, Sen Li, and Garrison W. Cottrell. 2019. Learning Representations for Time Series Clustering. *Advances in Neural Information Processing Systems* 32 (2019), 3781–3791.

[17] Naveen Sai Madiraju, Seid M. Sadat, Dimitry Fisher, and Homa Karimabadi. 2018. Deep Temporal Clustering: Fully Unsupervized Learning of Time-Domain Features. (2018).

[18] Pankaj Malhotra, Vishnu TV, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. 2017. Timenet: Pre-trained deep recurrent neural network for time series classification. *arXiv preprint arXiv:1706.08838* (2017).

[19] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems* 26 (2013), 3111–3119.

[20] John Paparrizos and Luis Gravano. 2015. k-Shape: Efficient and Accurate Clustering of Time Series. (2015), 69–76.

[21] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).

[22] Mingjie Qian and Chengxiang Zhai. 2013. Robust unsupervised feature selection. *Twenty-Third International Joint Conference on Artificial Intelligence* (2013).

[23] William M. Rand. 1971. Objective criteria for the evaluation of clustering methods. 66(336) (1971), 846–850.

[24] Hiroaki Sakoe and Seibi Chiba. 1978. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26 (1978), 43–49.

[25] Tim Salimans and Diederik P. Kingma. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in Neural Information Processing Systems* 29 (2016), 901–909.

[26] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. *IEEE Conference on Computer Vision and Pattern Recognition* (2015), 815–823.

[27] David Sculley. 2010. Web-scale k-means clustering. (2010), 1177–1178.

[28] Sohil Atul Shah and Vladlen Koltun. 2017. Deep Continuous Clustering. (2017).

[29] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. WaveNet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499* (2016).

[30] Gary M. Weiss, Kenichi Yoneda, and Thaier Hayajneh. 2019. Smartphone and Smartwatch-Based Biometrics Using Activities of Daily Living. *IEEE Access* (2019).

[31] Ledell Wu, Adam Fisch, Sumit Chopra, Keith Adams, Antoine Bordes, and Jason Weston. 2018. Starspace: Embed all the things! *Thirty-Second AAAI Conference on Artificial Intelligence* (2018).

[32] Lingfei Wu, Ian En-Hsu Yen, Jinfeng Yi, Fangli Xu, Qi Lei, and Michael Witbrock. 2018. Random Warping Series: A random features method for time-series embedding. 84 (2018), 793–802.

[33] Junyuan Xie, Ross Girshick, and Ali Farhadi. 2016. Unsupervized Deep Embedding for Clustering Analysis. *ICML* (2016), 198–202.

[34] Bo Yang, Xiao Fu, Nicholas D. Sidiropoulos, and Mingyi Hong. 2017. Towards K-means-friendly Spaces: Simultaneous Deep Learning and Clustering. *arXiv preprint arXiv:1610.04794v2* (2017).

[35] Jianwei Yang, Devi Parikh, and Dhruv Batra. 2016. Joint Unsupervized Learning of Deep Representations and Image Clusters. *arXiv preprint arXiv:1604.03628v3* (2016).

[36] Fisher Yu and Vladlen Koltun. 2016. Multi-scale context aggregation by dilated convolutions. *International Conference on Learning Representations* (2016).

[37] Qin Zhang, Jia Wu, Peng Zhang, Guodong Long, and Chengqi Zhang. 2018. Salient subsequence learning for time series clustering. (2018).