Unsupervised Scalable Representation Learning for Multivariate Time Series

Jean-Yves Franceschi, Aymeric Dieuleveut, Martin Jaggi

Unsupervised Scalable Representation Learning for Multivariate Time Series

Jean-Yves Franceschi [1 *] **Aymeric Dieuleveut** [2] **Martin Jaggi** [2]

Abstract

Time series constitute a challenging data type for machine learning algorithms, due to their highly variable lengths and sparse labeling in practice. In this paper, we tackle this challenge by proposing an unsupervised method to learn universal embeddings of time series. Unlike previous works, it is scalable with respect to their length and we demonstrate the quality, transferability and practicability of the learned representations with thorough experiments and comparisons. To this end, we combine an encoder based on causal dilated convolutions with a triplet loss employing time-based negative sampling, obtaining general-purpose representations for variable length and multivariate time series.

1. Introduction

We investigate in this work the topic of unsupervised general-purpose representation learning for time series. In spite of the increasing amount of work about representation learning in fields like natural language processing (Young et al., 2018) or videos (Denton & Fergus, 2018; Denton & Birodkar, 2017; Sermanet et al., 2017; Villegas et al., 2017; Goroshin et al., 2015; Srivastava et al., 2015; Tran et al., 2015), few articles explicitly deal with general-purpose representation learning for time series, and feature learning for time series without structural assumption on non-temporal data has received much less attention.

This problem is indeed challenging for various reasons. First, real-life time series are rarely or sparsely labeled. Therefore, *unsupervised* representation learning would be strongly preferred. Secondly, methods need to deliver compatible representations while allowing the input time series to have unequal lengths. Thirdly, scalability and efficiency both at training and inference time is crucial, in the sense that the techniques must work for both short and long time series encountered in real-world data.

Hence, we propose in the following an *unsupervised* method to learn *general-purpose representations* for *multivariate* time series that comply with the issues of *varying and potentially high lengths* of the studied time series. To this end, we adapt recognized deep learning tools and introduce a novel unsupervised loss. Our representations are computed by a deep convolutional neural network with dilated convolutions (Oord et al., 2016), preferred to recurrent neural networks for their scalability to long sequences, and adapted so that its output is a fixed-length vector, independent of the variable length of the input time series. This network is then trained unsupervised, using the first specifically designed triplet loss in the literature of time series, taking advantage of the encoder resilience to time series of unequal lengths.

We assess the quality of the learned representations on various datasets to ensure their universality. In particular, we test how our representations can be used for classification tasks on the standard datasets in the time series literature, compiled in the UCR repository (Dau et al., 2018). We show that our representations are *general* and *transferable*, and that the induced classification performance of our method *matches the state-of-the-art* of non-ensemble supervised classification techniques. However, these standard datasets are limited, as they only contain short univariate time series[1], whose lengths are constant across each dataset (except for a few ones). To overcome this limitation, we also evaluate our representations on the recently released UEA multivariate time series repository (Bagnall et al., 2018), as well as on a real-life dataset including very long time series, on which we demonstrate *scalability*, *performance* and generalization ability *across different tasks* beyond classification.

This paper is organized as follows. Section 2 exposes previous works on unsupervised representation learning and deep architectures for time series in the literature. Section 3 details the architecture of the encoder, while Section 4 describes how the encoder is trained. Finally, Section 5 provides results of the experiments that we conducted to evaluate our method.

[*]Work partially done while studying at ENS de Lyon and MLO, EPFL. [1]Sorbonne Université, CNRS, Laboratoire d'informatique de Paris 6, LIP6, F-75005 Paris, France [2]MLO, EPFL, Lausanne CH-1015, Switzerland. Correspondence to: Jean-Yves Franceschi <jean-yves.franceschi@ens-lyon.org>.

[1]Their lengths vary from 24 to 3 000, with most of them shorter than 1 000.

2. Related Work

To our knowledge, few recent works deal with unsupervised representation learning for time series. Hyvarinen & Morioka (2016) learn representations on evenly sized subdivisions of time series by learning to discriminate between those subdivisions from these representations. Lei et al. (2017) expose an unsupervised method designed so that the distances between learned representations mimic a standard distance (Dynamic Time Warping, DTW) between time series. Malhotra et al. (2017) design an encoder as a recurrent neural network, jointly trained with a decoder as a sequence-to-sequence model to reconstruct the input time series from its learned representation. Finally, Wu et al. (2018a) compute feature embeddings generated in the approximation of a carefully designed and efficient kernel.

However, these methods either are not scalable nor suited to long time series (due to the sequential nature of a recurrent network, or to the use of DTW with a quadratic complexity with respect to the input length), are tested on no or very few standard datasets, or do not provide sufficient comparison to assess the quality of the learned representations. Our model and analysis aim at overcoming these issues.

Deep convolutional neural networks have recently been used to handle time series in classification tasks (Cui et al., 2016; Wang et al., 2017), showing competitive performance. Dilated convolutions, popularized by WaveNet (Oord et al., 2016) for audio generation, have been used to improve their performance and were shown to perform well as sequence-to-sequence models for time series forecasting (Bai et al., 2018) using an architecture that inspired ours. These works particularly show that dilated convolutions help to build networks for sequential tasks that are able to outperform recurrent neural networks in terms of both efficiency and performance.

3. Encoder Architecture

Our choice of architecture for the encoder network is motivated by three requirements:

- it must extract relevant information from time series;
- it needs to be time- and memory-efficient, both for training and testing;
- it has to allow variable-length inputs.

We present and explain this choice in this section.

3.1. Overall Idea and Dilated Convolutions

We choose to use deep convolutional neural networks with dilated convolutions to handle time series. Compared to recurrent neural networks, which are inherently designed for sequence-modeling tasks and thus sequential, these networks are efficient as they allow efficient parallelization on modern hardware such as GPUs. We create an adapted version of the causal sequence-to-sequence model introduced by Bai et al. (2018) for forecasting tasks, augmented by a global max pooling layer to squeeze the temporal dimension, and followed by a fully connected layer to output the final representation.

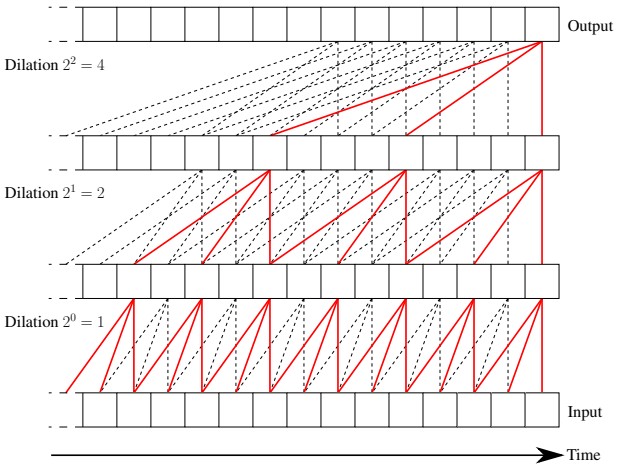


Figure 1. Illustration of stacked dilated causal convolutions. Lines between each sequence represent the computational graph of three stacked dilated causal convolutions. Red solid lines highlight the dependency graph for the computation of the last value of the output sequence, showing that no future value of the input time series is used to compute it.

This causal model consists of stacks of exponentially *dilated causal* convolutions (see Figure 1 for an illustration). Each convolutional layer maps, using appropriate padding, a sequence to a sequence of the same length, such that the i-th element of the output sequence is computed using only values up until the i-th element of the input sequence, for all i; thus, it is called *causal*, since the output value corresponding to a given time step is not computed using future input values. If stacked, these convolutions shape a network which retains the same property. In order to increase the width of the receptive field for the computation of any sequence value while keeping an efficient and scalable architecture, stacked convolutional layers have an *exponentially increasing dilation parameter*.

We detail this architecture and its characteristics in the remaining of this section.

3.2. Description

A causal convolution C with kernel size k and dilation parameter d is defined in the one-dimensional case (the

generalization to the multi-channel case is straightforward; we ignore it for the sake of clarity) as:

$$C : x \mapsto \left(\sum_{i=0}^{k-1} c_i x_{j-di} \right)_{j \in [\![(k-1)d, \text{size}(x)-1]\!]},$$

where $(c_i)_{i \in [\![0,k-1]\!]}$ are the convolution parameters. A fully connected causal convolution is retrieved when $d = 1$. When $d \neq 1$, the computation of $C(x)_j$ is done using inputs elements placed at distances from x_j that are multiples of d. By padding the input of C to the left with $(k-1)\,d$ elements, C becomes a causal convolution mapping a sequence to a sequence of the same length; in practice, we use zero-padding. From now on in this article, a causal convolution refers to the previous ensemble of a causal dilated convolution augmented with the previously described padding.

Following the recommendations of Bai et al. (2018), we build each layer of our network to be the succession of a causal convolution, weight normalization (Salimans & Kingma, 2016), leaky ReLU, another causal convolution with the same kernel size and dilation parameter as the first one, weight normalization and leaky ReLU, augmented with a residual connection from the input of the layer (see Figure 2a). If the input and output sequences of the layer do not have the same dimension, the residual connection is done using an upsampling (or downsampling) convolution with kernel size 1 and dilation size 1. Notice that all operations in this layer are causal, thus this layer and the succession of such layers are also causal.

The final architecture of our encoder is then formed by a succession of a given number of layers with exponentially increasing dilation parameters (the i-th layer is given dilation parameter 2^i) mapping the input sequence to a sequence of the same length in a causal manner, a global softmax pooling layer that squeezes the temporal dimension and aggregates all temporal information in a fixed-size vector (as proposed by Wang et al. (2017) in a supervised setting with full convolutions), and finally a fully-connected linear layer whose output, which is also fixed-length, is the output of the encoder. See Figure 2b for an illustration.

3.3. Further Motivation for Causal Dilated Convolutions

Efficiency. Besides their demonstrated efficiency on modern hardware, exponentially dilated convolutions have also been introduced to increase the ability of convolutional networks to better capture long-range dependencies by exponentially increasing the receptive field of the network at the same depth level of the network (Oord et al., 2016; Yu & Koltun, 2016; Bai et al., 2018).

Additionally, using a causal convolutional network followed by a global max pooling layer can alleviate the disadvantage of not using recurrent networks at testing time. Indeed, recurrent networks can be used in an online fashion, thus saving memory and computation time during testing. In our architecture, adding an element to the input time series does not require to evaluate the whole network on the input, as it would be the case with non-causal convolutions: it suffices to compute its corresponding output in the causal network with the computation graph highlighted in Figure 1, and update the outputs of the global max pooling and fully connected layers accordingly.

Performance. Recurrent networks are known to be subject to the issue of exploding and vanishing gradients, due to their recurrent nature (Goodfellow et al., 2016, Chapter 10.9). While significant work has been done to tackle it and improve their ability to capture long-term dependencies, such as the LSTM (Hochreiter & Schmidhuber, 1997), recurrent networks are still outperformed by convolutional networks on this aspect (Bai et al., 2018).

On the specific domain of time series classification, which is an essential part of our experimental evaluation, deep neural networks have not shown significant results compared to the state of the art of the domain for a long time (Bagnall et al., 2017), but have recently been successfully used thanks to convolutional networks (Cui et al., 2016; Wang et al., 2017).

4. Unsupervised Training

We explain in this section how the previously presented network can be trained in an unsupervised fashion.

We choose to use a triplet loss, inspired by approaches used in the field of word representation learning with word2vec (Mikolov et al., 2013), but also in other domains as well (Wang et al., 2014; Wu et al., 2018b). The objective is to ensure that similar time series obtain similar representations, with no supervision to learn such similarity. The assumption made in word2vec is twofold. The representation of a word should be, on one hand, close to the one of its *context* (Goldberg & Levy, 2014), i.e., the distribution of words which they have have been associated to in sentences, and, on the other hand, distant from the one of randomly chosen contexts, since they are probably different than the original word's context. The corresponding loss then pushes pairs of (word, context) and (word, random context) to be linearly separable. This technique is called *negative sampling*.

To adapt this principle to time series, we consider a random subseries[2] x^{ref} of a given time series y_i. Then, on one hand, its representation should be close to the one of any of its own subseries x^{pos} (a *positive* example). On the other hand, if we

[2] I.e., a subsequence of a time series composed by consecutive time steps of this time series.

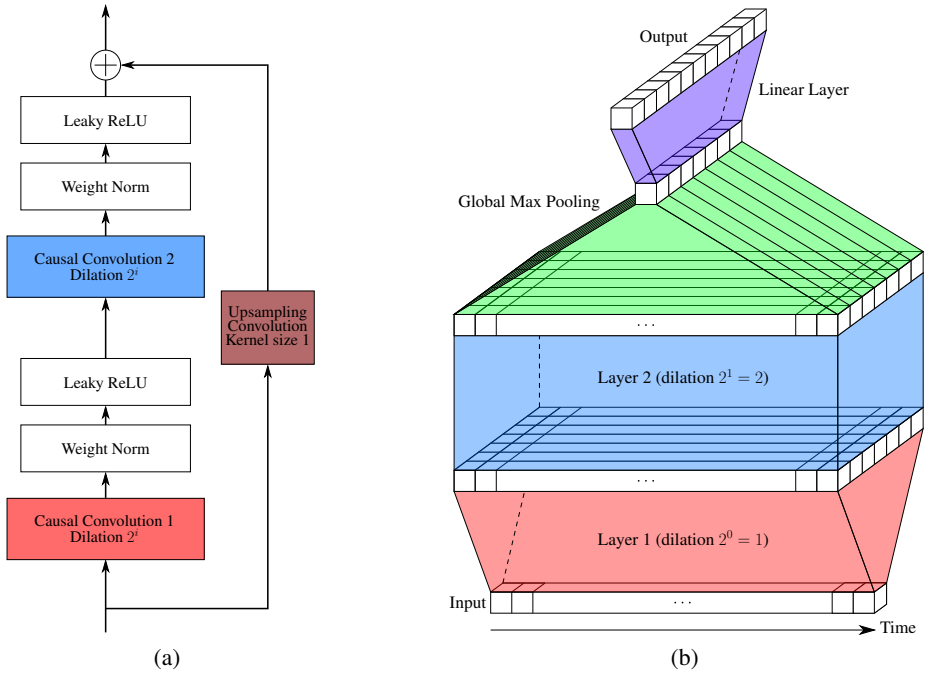


Figure 2. (a) Composition of the i-th layer of the chosen architecture, when the number of input channels is smaller than the number of output channels. (b) Example of the whole encoder architecture with two causal convolution layers.

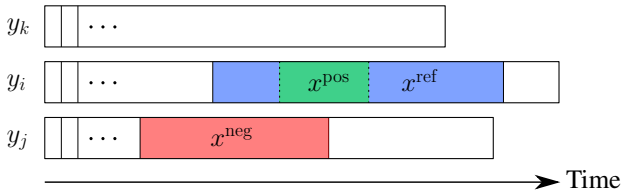


Figure 3. Illustration of the choices of x^{ref}, x^{pos} and x^{neg}.

consider another subseries x^{neg} (a *negative* example) chosen at random (in the same time series if it is long enough, or in a different random time series y_j if a dataset is available), then its representation should probably be distant from the one of x^{ref}. See Figure 3 for an illustration. Following the comparison with word2vec, x^{pos} corresponds to a word, x^{ref} to its context, and x^{neg} to a random context. The objective to be minimized corresponding to these choices, similarly to the one of word2vec with its shallow network replaced by a deep network $f(., \theta)$ with parameters θ, is:

$$-\log\left(\sigma\left(f\left(x^{\text{ref}}, \theta\right)^{\top} f\left(x^{\text{pos}}, \theta\right)\right)\right) - \log\left(\sigma\left(-f\left(x^{\text{ref}}, \theta\right)^{\top} f\left(x^{\text{neg}}, \theta\right)\right)\right), \quad (1)$$

where σ is the sigmoid function. This loss pushes the computed representations to distinguish between x^{ref} and x^{neg}, and to assimilate x^{ref} and x^{pos}. Overall, the training procedure consists in traveling through the training dataset for several epochs (possibly using mini-batches), picking tuples $\left(x^{\text{ref}}, x^{\text{pos}}, x^{\text{neg}}\right)$ at random, and performing a minimization step on the corresponding loss for each pair, until training ends.

To improve the stability and convergence of the training procedure as well as the experimental results of our learned representations, we allow, as in word2vec, our loss to take into account several negative samples $(x_k^{\text{neg}})_{k \in [\![1,K]\!]}$, chosen independently at random:

$$-\log\left(\sigma\left(f\left(x^{\text{ref}}, \theta\right)^{\top} f\left(x^{\text{pos}}, \theta\right)\right)\right) - \sum_{k=1}^{K} \log\left(\sigma\left(-f\left(x^{\text{ref}}, \theta\right)^{\top} f\left(x_k^{\text{neg}}, \theta\right)\right)\right). \quad (2)$$

In practice, we pick tuples $\left(x^{\text{ref}}, x^{\text{pos}}, (x_k^{\text{neg}})_{k \in [\![1,K]\!]}\right)$ in the following manner. We iterate over the available dataset for a given number of epochs (given as hyperparameter). For each train time series z, the length of x^{pos} is chosen uniformly at random in $[\![1, \text{size}(z)]\!]$; then the size of x^{ref} is chosen uniformly at random in $[\![\text{size}(x^{\text{pos}}), \text{size}(z)]\!]$, and x^{ref} is chosen uniformly at random among all subseries of z of the chosen size. Similarly, x^{pos} is chosen uniformly at random in x^{ref}. The choice of $(x_k^{\text{neg}})_{k \in [\![1,K]\!]}$ consists in simply choosing uniformly at random the time series which they will be drawn from, then their length, then picking them

at random as well according to those chosen parameters. The generalization of this procedure to mini-batch training is straightforward, so we do not detail it.

The length of the negative examples can either be the same for all samples and equal to $\text{size}\,(x^{\text{pos}})$, or be chosen at random similarly to $\text{size}\,(x^{\text{pos}})$. The first case is suitable when all time series in the dataset have equal lengths, and speeds up the training procedure thanks to computation factorizations; the second case is only used when time series in the dataset do not have the same lengths, as we saw no other difference than time efficiency between the two cases in our experiments.

We highlight that this training procedure takes advantage of the ability of the chosen encoder to take as input time series of different lengths. By training the encoder on a range of input lengths going from one to the length of the longest time series in the training set, it becomes able to output meaningful representations regardless of the input length, as shown in Section 5.

This training procedure is interesting in that it is efficient enough to be run over long time series (see Section 5), thanks to the efficiency of the chosen architecture, and of the separability of the loss, on which a backpropagation per term can be performed to save memory. Moreover, it only requires to train an encoder network, while standard representations learning methods jointly train an encoder and decoder in an autoencoder framework, as done by Malhotra et al. (2017), which induces a larger computational cost. As far as we know, this work is the first in the time series literature to propose a triplet loss for feature learning.

5. Experimental Results

We review in this section experiments conducted to investigate the relevance of the representations we learn. Code corresponding to these experiments is publicly available[3].

5.1. Classification

We first assess the *quality* of our learned representations by using them for time series classification, on which we obtain close to state-of-the-art results. We also highlight the *transferability* of our representations.

5.1.1. Protocol and Implementation Details

For each considered dataset with a train / test split, we unsupervisedly train an encoder using its train set. We then train an SVM with radial basis function kernel on top of the learned features using the training labels of the given dataset, and output the corresponding classification score on the test set. As our training procedure encourages representations of different time series to be separable, observing the classification performance of a simple SVM on these features is a good mean to check their quality.

We found that the number of negative samples to draw at each step of the encoder learning stage can have a significant impact on the performance of the encoder. E.g., for univariate time series classification, we trained for each dataset four encoders, one for each number of drawn negative samples $K \in \{1, 2, 5, 10\}$, and present results for each one of them. We also present a *combined* version, where all four pairs of encoders and SVMs are put together in a voting classifier, outputting the class that gets the majority of the predictions among the four classifiers. This enables our learned representations with different parameters to complement each other, and to remove some random noise in the classification scores.

We perform no hyperparameter optimization for the training of the encoder, except for the number of epochs which is dynamically tuned by an early stopping heuristic, only using the training labels as additional information. We use it for two reasons: as the same hyperparameters are used for all datasets, we chose the number of epochs to be large; early stopping then prevents overfitting due to this parameter and saves computation time. Note that, even if it introduces supervision in the encoder training, it is strictly equivalent to a more computationally demanding and fully unsupervised training, and remains optional, as discussed in the supplementary material, Section S1.2.

The full training process and hyperparameter choices are detailed in the supplementary material, Sections S1.1 and S2. We used Python 3 for implementation, with PyTorch 0.4.1 (Paszke et al., 2017) for neural networks and scikit-learn (Pedregosa et al., 2011) for SVMs. Each encoder was trained on a single Nvidia Titan Xp GPU with CUDA 9.0.

5.1.2. Univariate Time Series

We present accuracy scores for all datasets of the new iteration of the UCR archive (Dau et al., 2018), which is a standard set of varied univariate datasets. We preprocess datasets of the archive that were not already normalized so that the set of time series values for each dataset has zero mean and unit variance.

We compare our scores to the ones of the four best classifiers of the state-of-the-art presented in Bagnall et al. (2017)[4]: COTE (Bagnall et al., 2015), ST (Bostrom & Bagnall, 2015), BOSS (Schäfer, 2015) and EE (Lines & Bagnall, 2015), on the first 85 datasets of the archive[5]. We also add DTW

[3] https://github.com/White-Link/UnsupervisedScalableRepresentationLearningTimeSeries.

[4] http://www.timeseriesclassification.com/singleTrainTest.csv.

[5] The new UCR archive includes 43 new datasets on which no reproducible results of state-of-the-art methods have been pro-

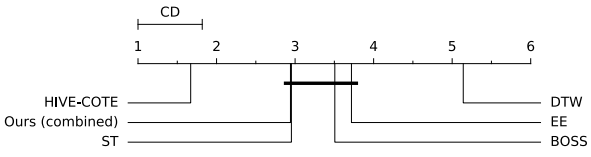


Figure 4. Critical difference diagram of the average ranks of the compared classifiers for the Nemenyi test, obtained with Orange (Demšar et al., 2013).

(which is a one-nearest neighbor classifier with DTW as metric) as a baseline to the comparison.

While DTW is an *unsupervised* method, the other ones are *supervised*. COTE is a powerful ensemble method using many classifiers on several representations of time series (such as Fourier); EE is a simpler ensemble method. ST and BOSS are supervised[6] as well: the former is based on shapelets and the latter is a dictionary-based classifier. Note that instead of reporting results for COTE, we report results for HIVE-COTE (Lines et al., 2018), which is a refinement of COTE and has been found to outperform it. HIVE-COTE assembles classifiers in a hierarchical voting structure to take advantage of their respective strengths. In particular, it includes ST, BOSS and EE in its ensemble, and is thus expected to outperform them.

Methods based on neural networks have emerged recently (Cui et al., 2016; Wang et al., 2017) and claim to achieve close to state-of-the-art performance, but, to our knowledge, there was no extensive study similar to Bagnall et al. (2017) comparing them in similar settings and in a reproducible manner. We provide a comparison of our method with the reported scores of FCN in Wang et al. (2017), who was only tested on half the old UCR archive, in the supplementary material, Section S3, Table S3. We provide there as well comparisons with TimeNet (Malhotra et al., 2017) and RWS (Wu et al., 2018a), which are two unsupervised methods also training a simple classifier on top the learned representations, and reporting their results on a few UCR datasets.

We report in Table 1 scores for only some UCR datasets, while scores for all datasets are reported in the supplementary material, Section S3.

Performance. For an overall analysis of the performance of our method, we show in Figure 4 the critical difference diagram of all compared methods, as in Bagnall et al. (2017). It shows that our method is globally second-to-best, only beaten by HIVE-COTE and equivalent to ST. Thus, our unsupervised method beats several recognized supervised classification algorithms, and is only preceded by a powerful ensemble method, which was expected since it takes advantage of the numerous classifiers and data representation it uses. More details are presented in Figure 5: the histogram of ranks shows that our method ranks first 22% of the time (versus 55% for HIVE-COTE and 25% for ST), and is in the top 3 60% of the time (versus 96% for HIVE-COTE and 66% for ST), while the boxplot shows that our method has second-to-best median regarding the ratio of accuracy over maximum achieved accuracy, behind HIVE-COTE and above ST. Its limitation comes from the fact that, for some datasets, it performs significantly worse than other methods: it corresponds to cases where our encoder could not uncover the differences between differently labeled time series, which is expected due to its unsupervised training. Overall, our method, while expectedly beaten by HIVE-COTE, matches the second-to-best studied supervised method, and in particular is *at the level of the best performing method included in HIVE-COTE*[7].

Note that partial results also indicate that our method is beaten (on 68% out of 44 UCR datasets) by FCN, which are neural network trained for supervised classification, and thus expected to beat our neural network trained unsupervisedly (see the supplementary material, Section S3, Table S3). These results also indicate that our method *consistently outperforms both unsupervised methods* TimeNet and RWS (on, respectively, 13 and 9 out of 13 and 12 UCR datasets), showing its performance.

Transferability. We include in the comparisons the classification accuracy for each dataset of an SVM trained on this dataset using the representations computed by an encoder, which was trained *on another dataset* (FordA, with $K = 5$), to test the transferability of our representations.

We observe that the scores achieved by this SVM trained on transferred representations are close to the scores reported when the encoder is trained on the same dataset as the SVM, showing the *transferability* of our representations from a dataset to another, and from time series to other time series *with different lengths*. More generally, this transferability and the performance of simple classifiers on the representations we learn indicate that they are *universal* and *easy to make use of*.

5.1.3. Multivariate Time Series

We tested our method on all 30 datasets of the newly released UEA archive (Bagnall et al., 2018). For each dataset,

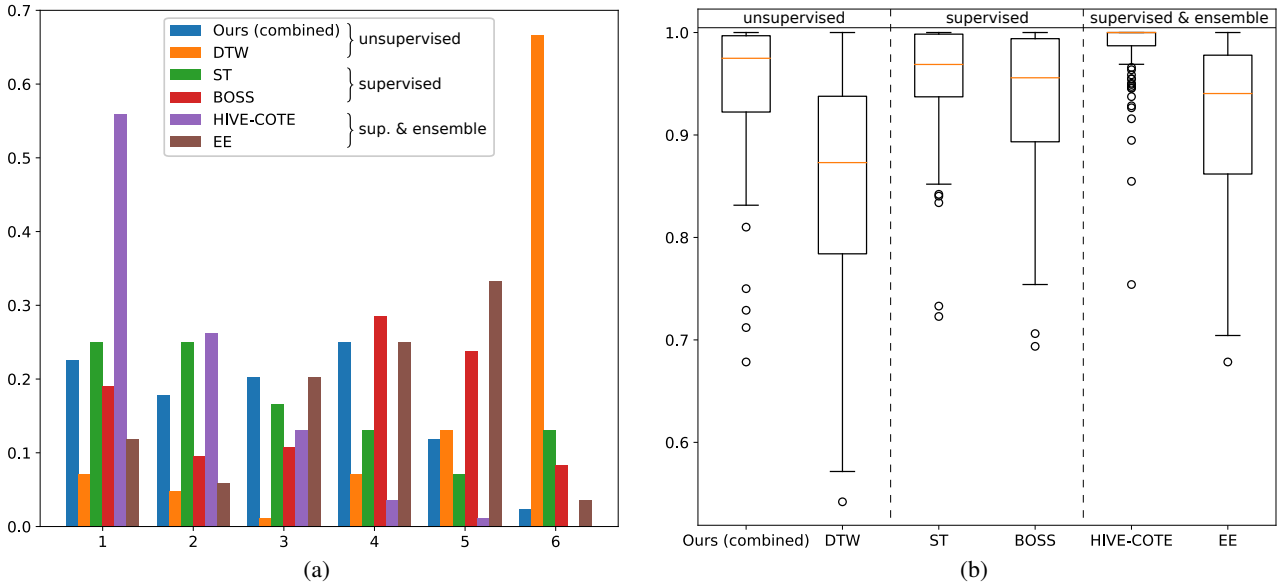
duced yet. Nevertheless, we provide complete results for our method on these remaining datasets as well, in the supplementary material, Section S3, Table S4, and only compare them to DTW.

[6]While ST and BOSS also use ensembles of classifiers, these classifiers are simple and applied on a specifically-designed representation of the input. Thus we do not qualify them as ensemble.

[7]It is possible to incorporate our representations in HIVE-COTE thanks to its flexibility. We suppose that it could improve its performance, but this is beyond the scope of this work.

Table 1. Accuracy scores of variants of our method compared with those of DTW (unsupervised), ST and BOSS (supervised) and HIVE-COTE and EE (supervised ensemble methods), on some UCR datasets. Results for the whole archive are available in the supplementary material, Section S3, Tables S1, S2 and S4. Bold scores indicate the best performing method.

Dataset	Ours (unsupervised)						Unsup.	Supervised		Supervised ensemble	
	$K = 1$	$K = 2$	$K = 5$	$K = 10$	Combined	FordA ($K = 5$)	DTW	ST	BOSS	HIVE-COTE	EE
DiatomSizeReduction	0.987	0.977	**0.993**	**0.993**	0.99	0.958	0.967	0.925	0.931	0.941	0.944
ECGFiveDays	**1**	**1**	**1**	**1**	**1**	0.768	**1**	0.984	**1**	**1**	0.82
FordB	0.767	0.784	0.774	0.788	0.798	0.764	0.62	0.807	0.711	**0.823**	0.662
Ham	0.638	0.552	**0.724**	0.6	0.657	0.723	0.467	0.686	0.667	0.667	0.571
Phoneme	0.251	0.253	0.248	0.255	0.272	0.225	0.228	0.321	0.265	**0.382**	0.305
SwedishLeaf	0.923	0.916	0.933	0.912	0.939	0.909	0.792	0.928	0.922	**0.954**	0.915



(a) (b)

Figure 5. (a) Distribution of the ranks of compared methods on the first 85 UCR datasets. (b) Boxplot of the ratio of the accuracy versus maximum achieved accuracy for compared methods on the first 85 UCR datasets.

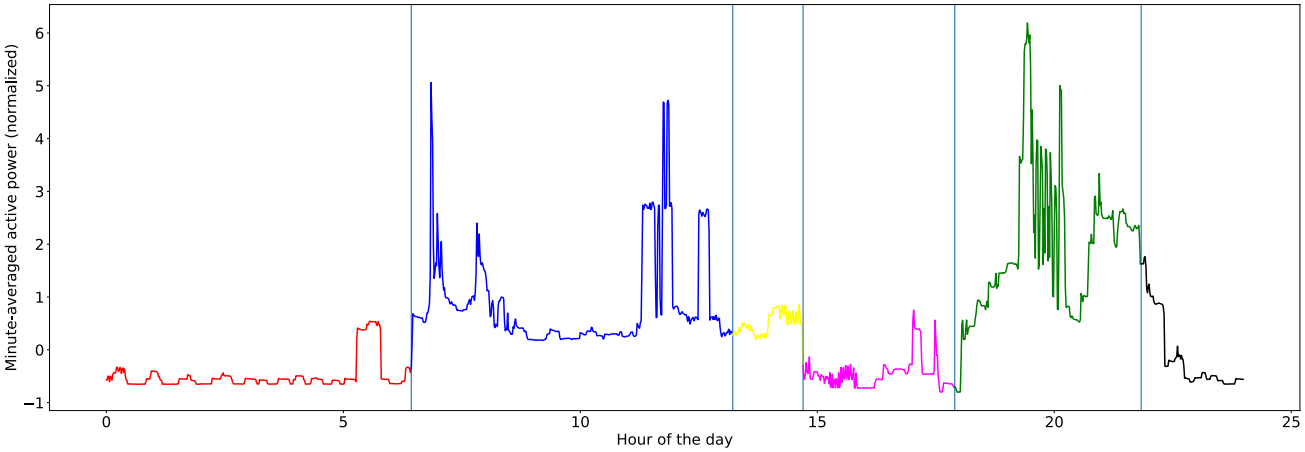


Figure 6. Minute-averaged electricity consumption for a single day, with respect to the hour of the day. Vertical lines and colors divide the day into six clusters, obtained with k-means clustering based on representations computed on a day-long sliding window. The clustering divides the day in meaningful portions (night, morning, afternoon, evening).

each dimension of the time series was preprocessed independently from the other dimensions by normalizing its mean and variance. Full accuracy scores are presented in the supplementary material, Section S4, Table S5.

The UEA archive has been designed as a first attempt to provide a standard archive for multivariate time series classification such as the UCR one for univariate series. As it has only been released recently, we could not compare our method to state-of-the-art classifiers for multivariate time series. However, we provide a comparison with DTW_D as baseline using results provided by Bagnall et al. (2018). DTW_D (dimension-Dependent DTW) is a possible extension of DTW in the multivariate setting, and is the best baseline studied by Bagnall et al. (2018).

Overall, our method matches or outperforms DTW_D on 72% of the UEA datasets, which indicates a good performance. As this archive is destined to grow and evolve in the future, and without further comparisons, no additional conclusion can be drawn.

5.2. Evaluation on Long Time Series

In this section, we show the *applicability* and *scalability* of our method on *long* time series without labeling for regression tasks, which could correspond to an industrial application.

The Individual Household Electric Power Consumption dataset from the UCI Machine Learning Repository (Dheeru & Karra Taniskidou, 2017) consists in a minute-averaged electricity consumption of a single household in France for four years. It thus corresponds to a single time series of length $2\,075\,259$.

We split this time series into train (first $500\,000$ measurements, corresponding to approximately a year) and test (last $1\,575\,259$ measurements), and normalize it to a zero-mean and unit variance time series. The encoder, whose hyperparameters are detailed in the supplementary material (Section S2), is trained over the train time series on a single Nvidia Tesla P100 GPU in no more than a few hours, showing that our training procedure is *scalable* to long time series.

We consider the learned encoder, trained on a year-long time series, on two regression tasks involving two different input scales. We compute, for each time step of the time series, the representations of the last window corresponding to a day ($1\,440$ measurements) and a quarter ($12 \cdot 7 \cdot 1\,440$ measurements)[8]. An example of application of the day-long representations is shown on Figure 6. The considered tasks consist in, for each time step, predicting the discrepancy of the mean value of the series during the next day (respectively, quarter) compared to the mean value of the previous day (respectively, quarter), given the representations computed on this day (respectively, quarter).

[8]Note that representations for both scales are computed with the same encoder.

Table 2. Results obtained on the Individual Household Electric Power Consumption dataset.

Task	Metric	Representations	Raw values
Day	Test MSE	$8.95 \cdot 10^{-2}$	$\mathbf{8.92 \cdot 10^{-2}}$
	Wall time	**12s**	3min 1s
Quarter	Test MSE	$7.81 \cdot 10^{-2}$	$\mathbf{6.26 \cdot 10^{-2}}$
	Wall time	**4s**	1h 40min 15s

We consider simple linear regressors over the learned representations, trained on the train time series using gradient descent to minimize the mean squared error between the prediction and the target. We compare their final scores on the test time series to the ones of linear regressors taking as input the raw values of the last day (respectively, quarter), trained in a similar manner.

Results and execution times on a Nvidia Titan Xp GPU are presented in Table 2. While regressors trained on our representations are *slightly less performing* than the ones trained on the raw time series values, they are *trained much more efficiently*, as they operate on small input sizes. Moreover, a single encoder trained with our procedure is able to output representations for different scales of input lengths that are helpful for other tasks than classification, corroborating their *universality*.

6. Conclusion

We presented an unsupervised general-purpose representation learning method for time series that is scalable and produces high-quality and easy-to-use embeddings. They are generated by an encoder formed by dilated convolutions that admits variable-length inputs, and trained with a novel triplet loss using adapted negative sampling for time series. The architecture of the encoder and the simplicity of the loss ensure the efficiency of the learned representations. Conducted experiments show that these representations are universal and can easily and efficiently be used for diverse tasks, especially for classification for which training SVMs on them produces close to state-of-the-art results in the domain, but also for regression. We leave as future work the applicability of our method to other tasks like forecasting, and the study of its impact if it were to be added in powerful ensemble methods.

7. Acknowledgements

We would like to acknowledge Patrick Gallinari, Sylvain Lamprier, Mehdi Lamrayah, Sidak Pal Singh, Andreas Hug, Jean-Baptiste Cordonnier and François Fleuret for helpful comments and discussions at various stages of this project.

References

Bagnall, A., Lines, J., Hills, J., and Bostrom, A. Time-series classification with COTE: The Collective of Transformation-Based Ensembles. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2522–2535, September 2015.

Bagnall, A., Lines, J., Bostrom, A., Large, J., and Keogh, E. 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, May 2017.

Bagnall, A., Dau, H. A., Lines, J., Flynn, M., Large, J., Bostrom, A., Southam, P., and Keogh, E. The UEA multivariate time series classification archive, 2018. *arXiv preprint arXiv:1811.00075*, 2018.

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

Bostrom, A. and Bagnall, A. Binary shapelet transform for multiclass time series classification. In *Big Data Analytics and Knowledge Discovery*, pp. 257–269, Cham, 2015. Springer International Publishing. ISBN 978-3-319-22729-0.

Cui, Z., Chen, W., and Chen, Y. Multi-scale convolutional neural networks for time series classification. *arXiv preprint arXiv:1603.06995*, 2016.

Dau, H. A., Keogh, E., Kamgar, K., Yeh, C.-C. M., Zhu, Y., Gharghabi, S., Ratanamahatana, C. A., Yanping, Hu, B., Begum, N., Bagnall, A., Mueen, A., and Batista, G. The UCR time series classification archive, October 2018.

Demšar, J., Curk, T., Erjavec, A., Črt Gorup, Hočevar, T., Milutinovič, M., Možina, M., Polajnar, M., Toplak, M., Starič, A., Štajdohar, M., Umek, L., Žagar, L., Žbontar, J., Žitnik, M., and Zupan, B. Orange: Data mining toolbox in Python. *Journal of Machine Learning Research*, 14: 2349–2353, 2013.

Denton, E. and Fergus, R. Stochastic video generation with a learned prior. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1174–1183. PMLR, July 2018.

Denton, E. L. and Birodkar, V. Unsupervised learning of disentangled representations from video. In *Advances in Neural Information Processing Systems 30*, pp. 4414–4423. Curran Associates, Inc., 2017.

Dheeru, D. and Karra Taniskidou, E. UCI machine learning repository, 2017.

Goldberg, Y. and Levy, O. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Goroshin, R., Mathieu, M. F., and LeCun, Y. Learning to linearize under uncertainty. In *Advances in Neural Information Processing Systems 28*, pp. 1234–1242. Curran Associates, Inc., 2015.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Hyvarinen, A. and Morioka, H. Unsupervised feature extraction by time-contrastive learning and nonlinear ICA. In *Advances in Neural Information Processing Systems 29*, pp. 3765–3773. Curran Associates, Inc., 2016.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Lei, Q., Yi, J., Vaculin, R., Wu, L., and Dhillon, I. S. Similarity preserving representation learning for time series analysis. *arXiv preprint arXiv:1702.03584*, 2017.

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

Lines, J., Taylor, S., and Bagnall, A. Time series classification with HIVE-COTE: The hierarchical vote collective of transformation-based ensembles. *ACM Transactions on Knowledge Discovery from Data*, 12(5):52:1–52:35, July 2018.

Malhotra, P., TV, V., Vig, L., Agarwal, P., and Shroff, G. TimeNet: Pre-trained deep recurrent neural network for time series classification. *arXiv preprint arXiv:1706.08838*, 2017.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pp. 3111–3119. Curran Associates, Inc., 2013.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. WaveNet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in PyTorch. 2017.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Salimans, T. and Kingma, D. P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems 29*, pp. 901–909. Curran Associates, Inc., 2016.

Schäfer, P. The BOSS is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, 29(6):1505–1530, Nov 2015.

Sermanet, P., Lynch, C., Chebotar, Y., Hsu, J., Jang, E., Schaal, S., and Levine, S. Time-contrastive networks: Self-supervised learning from video. *arXiv preprint arXiv:1704.06888*, 2017.

Srivastava, N., Mansimov, E., and Salakhudinov, R. Unsupervised learning of video representations using LSTMs. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 843–852, Lille, France, July 2015. PMLR.

Tran, D., Bourdev, L., Fergus, R., Torresani, L., and Paluri, M. Learning spatiotemporal features with 3D convolutional networks. In *IEEE International Conference on Computer Vision*, pp. 4489–4497, December 2015.

Villegas, R., Yang, J., Hong, S., Lin, X., and Lee, H. Decomposing motion and content for natural video sequence prediction. *International Conference on Learning Representations*, 2017.

Wang, J., Song, Y., Leung, T., Rosenberg, C., Wang, J., Philbin, J., Chen, B., and Wu, Y. Learning fine-grained image similarity with deep ranking. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1386–1393, June 2014.

Wang, Z., Yan, W., and Oates, T. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 1578–1585, May 2017.

Wu, L., Yen, I. E.-H., Yi, J., Xu, F., Lei, Q., and Witbrock, M. Random Warping Series: A random features method for time-series embedding. In *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pp. 793–802. PMLR, April 2018a.

Wu, L. Y., Fisch, A., Chopra, S., Adams, K., Bordes, A., and Weston, J. Starspace: Embed all the things! In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018b.

Young, T., Hazarika, D., Poria, S., and Cambria, E. Recent trends in deep learning based natural language processing. *IEEE Computational Intelligence Magazine*, 13(3):55–75, August 2018.

Yu, F. and Koltun, V. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations*, 2016.

Appendices

In these appendices, we provide our detailed training procedure for classification tasks, choices of hyperparameters, as well as the full experimental results of our method, compared to other concurrent methods. Section S1 explains and discusses the exact early stopping strategy and SVM training used for classification tasks, and discusses the importance of early stopping in the training of otherwise fully unsupervised representations. Secion S2 details the choices of hyperparameters in all presented experiments. Section S3 reports accuracy scores of all variants of our method on the whole UCR archive (Dau et al., 2018), as well as comparisons with concurrent methods, when available. Finally, Section S4 provides accuracy scores for our method on the whole UEA archive (Bagnall et al., 2018).

S1. Detailed Training for Classification Tasks

S1.1. SVM Training and Early Stopping

We perform no hyperparameter optimization on the architecture of our encoder, nor on the batch size or optimizer we use. We thus perform a single training procedure for each dataset and parameter K. The only parameters we dynamically tune are the number of epochs to train the encoder through an early stopping heuristic (stop training after a given number of epochs have been done without increasing a performance score and until a given number of epochs is reached, and keep the encoder corresponding to the best score), and the penalty C of the error term of the SVM.

In order to tune the latter and monitor a performance test which is not the train classification score for the early stopping criterion, we use as performance score a cross-validation score on the training set in the following manner. To choose a penalty for the SVM, we pick the one that achieves the best cross-validation mean classification score on the representations of the train set. The performance monitored at the end of each training epoch of the encoder is this cross-validation score for the best found SVM penalty, on the current representations of the train set. Note that if the train set or the number of training samples per class are too small, we do not use early stopping and choose a penalty $C = \infty$ for the SVM (which corresponds to no regularization).

This complex scheme is required because the avalaible UCR and UEA archive do not provide any additional validation set. Because lots of datasets are small, and to guarantee a fair comparison with concurrent methods which do not use any validation set, we designed the early stopping strategy to only use training labels.

S1.2. Early Stopping Discussion

With such an early stopping criterion, the entire method is then not fully unsupervised, because the labels are used to decide when to stop the learning procedure. This choice was mainly made to avoid having extra hyper-parameters to tune, and to save time on computation by avoiding a long training on some datasets. It does not change much the overall results, as it improves the accuracy on some datasets, but worsens them on some others. As an example, we provide in Figure S1, the evolution of the test accuracy with respect to the number of epochs, showing that the stopping time is not optimal. Besides, the encoder can always be trained without label information (stopping after a certain number of epochs), or with very sparsely labeled time series.

Moreover, this encoder training augmented with this early stopping criterion is strictly equivalent to a more computationally demanding and *fully unsupervised* training. Indeed, consider the training of the encoder for a number of epochs equal to the maximal number of epochs under the early stopping procedure. If one records the weights of the encoder at the end of each epoch, then one could simulate the online early stopping heuristic in an offline fashion by iteratively computing the early stopping performance score, stopping when the early stopping conditions are met and retain the best set of weights. This way, the encoder training is fully unsupervised at the cost of longer and more complex training using the train labels. Note that exploratory experiments indicate that selecting the best performing set of weights over the whole number of epochs, instead of simulating early stopping, tends to give results similar to the ones obtained with early stopping.

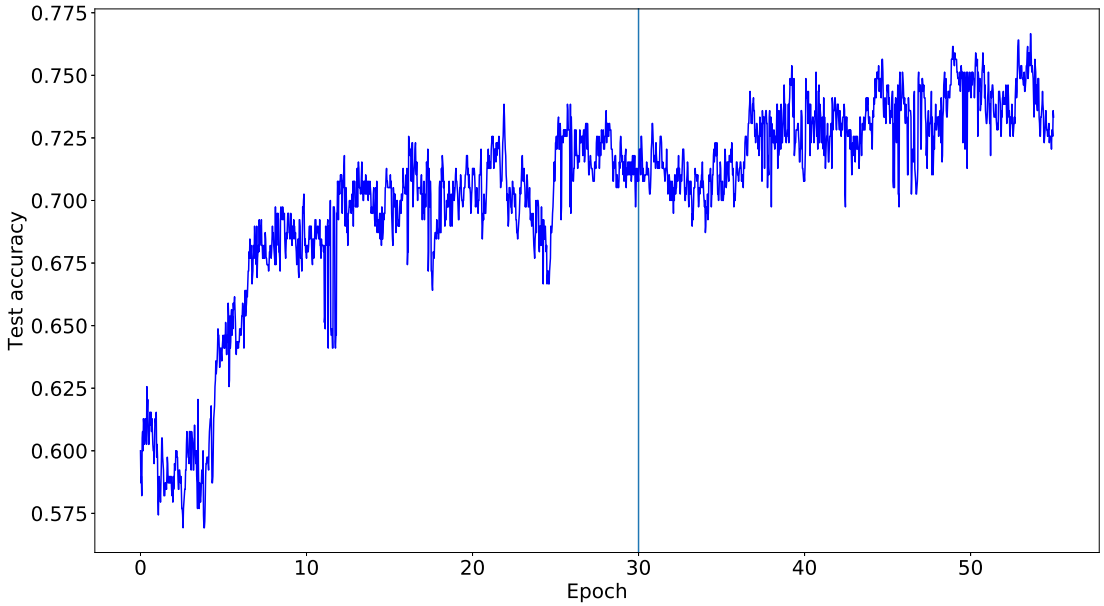


Figure S1. Evolution of the test accuracy during the training of the representation on the CricketX dataset from the UCR archive (with $K = 10$), with respect to the number of completed epochs. The test labels were only used for monitoring purposes and the test accuracy was computed after each mini-batch optimization. The vertical line marks the epoch selected by the early stopping heuristic. Test accuracy clearly increases during training, and the early stopping heuristic is suboptimal on this dataset.

S1.3. Behavior of the Learned Representations through Training

The *Risk* R is defined as the expectation (taken over the random selection of the sequences $\{x^{\mathrm{ref}}, x^{\mathrm{pos}}, x^{\mathrm{neg}}\}$) of the loss defined in Equation (1). This risk may decrease if all the representations $f(\cdot, \theta)$ are scaled by a positive large number. For example, if for some θ_0, for (almost surely) any sequences $\{x^{\mathrm{ref}}, x^{\mathrm{pos}}, x^{\mathrm{neg}}\}$, $f\left(x^{\mathrm{ref}}, \theta_0\right)^\top f\left(x^{\mathrm{pos}}, \theta_0\right) \geq 0$ and $f\left(x^{\mathrm{ref}}, \theta_0\right)^\top f\left(x^{\mathrm{neg}}, \theta_0\right) \leq 0$, then

$$R(\lambda, \theta_0) := \mathbb{E}_{x^{\mathrm{ref}}, x^{\mathrm{pos}}, x^{\mathrm{neg}}}\left[-\log\left(\sigma\left(\lambda^2 f\left(x^{\mathrm{ref}}, \theta_0\right)^\top f\left(x^{\mathrm{pos}}, \theta_0\right)\right)\right) - \log\left(\sigma\left(-\lambda^2 f\left(x^{\mathrm{ref}}, \theta_0\right)^\top f\left(x^{\mathrm{neg}}, \theta_0\right)\right)\right)\right] \quad (3)$$

is a decreasing function of λ, thus λ could diverge to infinity in order to minimize the loss. In other words, the parameters in θ_0 corresponding to the last linear layer could be linearly scaled up, and representations would "explode" (their norm would always increase through training). Such a phenomenon is not observed in practice, as the mean representation Euclidean norm lies around 20. There are two possible explanations for that: either the condition above is not satisfied (more generally, the loss is not reduced by increasing the representations) or the use of the sigmoid function, that has vanishing gradients, results in an increase of the representations that is too slow to be observed, or negligible with respect to other weight updates during optimization.

S2. Hyperparameters

We train our models with the following parameters for time series classification:

- optimizer: Adam (Kingma & Ba, 2014) with learning rate $\alpha = 0.001$ and decay rates $\beta = (0.9, 0.999)$;
- SVM: penalty $C \in \left\{10^i \mid i \in [\![-4, 4]\!]\right\} \cup \{\infty\}$;
- encoder training:
  - number of negative samples: $K \in \{1, 2, 5, 10\}$ for univariate time series, $K \in \{5, 10, 20\}$ for multivariate ones;
  - batch size: 10;
  - maximum number of epochs: 400;
  - number of epochs to wait without performance improvement for early stopping: 25;
- architecture:
  - number of channels in the intermediary layers of the causal network: 40;
  - number of layers (depth of the causal network): 10,
  - kernel size of all convolutions: 3;
  - negative slope of the leaky ReLU: 0.01;
  - number of output channels of the causal network (before max pooling): 320;
  - dimension of the representations: 160.

For the Individual Household Electric Power Consumption dataset, changes are the following:

- number of negative samples: $K = 10$;
- batch size: 1;
- no early stopping;
- number of channels in the intermediary layers of the causal network: 30;
- number of output channels of the causal network (before max pooling): 160;
- dimension of the representations: 80.

S3. Univariate Time Series

Full results corresponding to the first 85 UCR datasets for our method are presented in Table S1, while comparisons with DTW, ST, BOSS, HIVE-COTE and EE are shown in Table S2, and comparisons with FCN, TimeNet and RWS are shown in Table S3. Table S4 compiles the results of our method and of DTW[S1] for the newest 43 UCR datasets (except DodgerLoopDay, DodgerLoopGame and DodgerLoopWeekend which contain missing values).

All UCR datasets are provided with a unique train / test split that we used in our experiments. Compared techniques (DTW, ST, BOSS, HIVE-COTE and EE) were also tested on 100 random train / test splits of these datasets by (Bagnall et al., 2017) to produce a very strong state-of-the-art evaluation, but we did not perform similar resamples as this is beyond the scope of this work and would require much more computations. Note that the scores for these methods used in this article are the ones corresponding to the original train / test split of the datasets.

As our method is based on random sampling, the reported scores may vary depending on the random seed. While we do not report standard deviation, the large number of tested datasets prevents large statiscal error in the global evaluation of our method. The order of magnitude of accuracy variation between different runs of the combined version of our method is around 0.01 (for instance, for five different runs, the corresponding standard variations for, respectively, datasets Mallat, DiatomSizeReduction, CricketX and UWaveGestureLibraryX are 0.004, 0.014, 0.019 and 0.004).

[S1] Taken from https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.

Table S1: Accuracy scores of variants of our method on the first 85 UCR datasets. Bold scores indicate the best performing method.

Dataset	Ours (unsupervised)					
	$K = 1$	$K = 2$	$K = 5$	$K = 10$	Combined	FordA ($K = 5$)
Adiac	0.747	0.734	0.752	0.747	0.775	**0.788**
ArrowHead	0.777	0.8	0.834	0.766	**0.851**	0.783
Beef	0.633	0.667	0.633	0.633	0.633	**0.767**
BeetleFly	0.85	0.8	0.85	0.85	0.85	**0.95**
BirdChicken	0.8	**0.9**	0.75	**0.9**	**0.9**	**0.9**
Car	0.75	**0.867**	0.667	0.783	0.817	0.683
CBF	0.988	**0.999**	0.996	0.997	0.998	0.989
ChlorineConcentration	0.717	0.736	0.728	0.697	**0.75**	0.696
CinCECGtorso	0.641	0.733	**0.784**	0.699	0.726	0.63
Coffee	**1**	**1**	**1**	**1**	**1**	**1**
Computers	0.696	0.692	**0.708**	0.684	0.704	0.696
CricketX	0.751	0.715	0.715	0.751	**0.79**	0.618
CricketY	0.651	0.697	0.685	0.69	**0.726**	0.638
CricketZ	0.723	0.721	0.736	0.754	**0.767**	0.656
DiatomSizeReduction	0.987	0.977	**0.993**	**0.993**	0.99	0.958
DistalPhalanxOutlineCorrect	0.754	0.703	**0.775**	0.743	0.75	0.707
DistalPhalanxOutlineAgeGroup	0.727	0.734	**0.748**	0.741	**0.748**	0.741
DistalPhalanxTW	0.647	0.662	0.669	0.64	**0.676**	**0.676**
Earthquakes	**0.748**	**0.748**	**0.748**	**0.748**	**0.748**	**0.748**
ECG200	0.83	0.85	0.85	0.85	**0.87**	0.85
ECG5000	0.938	0.941	0.935	0.938	0.94	**0.943**
ECGFiveDays	**1**	**1**	**1**	**1**	**1**	**1**
ElectricDevices	0.722	0.712	0.687	0.682	**0.73**	0.676
FaceAll	0.782	0.761	0.757	0.779	**0.786**	0.75
FaceFour	0.795	0.841	0.886	**0.898**	0.875	0.864
FacesUCR	0.877	0.891	0.861	0.857	**0.893**	0.873
FiftyWords	0.745	0.752	0.767	0.723	**0.778**	0.668
Fish	0.909	0.874	0.874	0.811	0.897	**0.949**
FordA	0.924	0.92	0.922	0.92	**0.932**	0.922
FordB	0.767	0.784	0.774	0.788	**0.798**	0.764
GunPoint	0.993	0.98	0.993	0.987	0.987	**1**
Ham	0.638	0.552	**0.724**	0.6	0.657	0.723
HandOutlines	0.919	0.905	0.919	0.916	0.922	**0.924**
Haptics	0.474	0.484	0.471	0.494	**0.506**	0.494
Herring	**0.625**	0.563	0.578	0.469	0.609	0.594
InlineSkate	0.362	0.389	0.416	0.405	**0.418**	0.34
InsectWingbeatSound	0.565	0.586	0.583	**0.606**	0.603	0.566
ItalyPowerDemand	0.93	0.941	0.954	0.929	0.937	**0.96**
LargeKitchenAppliances	0.835	0.824	0.843	0.813	**0.861**	0.837
Lightning2	0.754	0.77	0.77	0.738	0.77	**0.836**
Lightning7	0.753	0.74	**0.822**	0.74	0.808	0.699
Mallat	0.934	**0.967**	0.943	0.957	0.962	0.909
Meat	0.933	0.917	0.917	0.917	0.917	**0.967**
MedicalImages	0.762	0.733	0.755	0.762	**0.778**	0.725
MiddlePhalanxOutlineCorrect	0.821	**0.842**	0.801	0.797	0.838	0.797
MiddlePhalanxOutlineAgeGroup	0.643	0.63	**0.656**	0.623	0.636	0.636
MiddlePhalanxTW	0.578	0.584	0.532	**0.591**	0.578	0.578
MoteStrain	0.846	**0.873**	0.847	0.863	0.859	0.864
NonInvasiveFatalECGThorax1	0.929	0.926	0.925	0.911	**0.938**	0.906

Table S1: Accuracy scores of variants of our method on the first 85 UCR datasets. Bold scores indicate the best performing method.

Dataset	Ours (unsupervised)					
	$K = 1$	$K = 2$	$K = 5$	$K = 10$	Combined	FordA ($K = 5$)
NonInvasiveFatalECGThorax2	0.933	0.935	0.938	0.936	**0.945**	0.916
OliveOil	0.867	**0.9**	0.833	0.867	**0.9**	0.867
OSULeaf	0.74	0.785	0.752	0.773	**0.814**	0.711
PhalangesOutlinesCorrect	0.789	0.79	0.762	0.798	0.808	**0.819**
Phoneme	0.251	0.253	0.248	0.255	**0.272**	0.225
Plane	**1**	**1**	**1**	**1**	**1**	0.99
ProximalPhalanxOutlineCorrect	0.863	**0.907**	0.859	0.866	0.89	0.904
ProximalPhalanxOutlineAgeGroup	0.844	0.844	0.829	**0.868**	0.839	0.844
ProximalPhalanxTW	0.78	0.79	0.78	0.776	**0.81**	0.771
RefrigerationDevices	0.541	**0.576**	0.493	0.461	0.515	0.528
ScreenType	0.485	0.475	0.411	0.483	**0.491**	0.411
ShapeletSim	0.606	0.557	**0.822**	0.639	0.75	0.644
ShapesAll	0.837	0.857	0.832	0.853	**0.865**	0.815
SmallKitchenAppliances	**0.733**	0.709	0.637	0.701	0.717	0.728
SonyAIBORobotSurface1	0.877	0.827	0.894	0.857	**0.895**	0.762
SonyAIBORobotSurface2	0.92	0.924	**0.941**	0.836	0.94	0.845
StarlightCurves	0.959	0.958	0.963	0.954	**0.964**	0.96
Strawberry	0.957	0.957	0.949	0.943	0.957	**0.962**
SwedishLeaf	0.923	0.916	0.933	0.912	**0.939**	0.909
Symbols	0.954	**0.96**	0.957	0.949	0.954	0.954
SyntheticControl	0.983	0.983	**0.993**	0.987	0.987	0.96
ToeSegmentation1	0.925	0.908	0.947	**0.961**	0.947	0.947
ToeSegmentation2	0.9	0.877	0.885	0.823	0.908	**0.938**
Trace	**1**	**1**	**1**	**1**	**1**	**1**
TwoLeadECG	**0.996**	0.985	0.975	**0.996**	0.989	0.98
TwoPatterns	**1**	**1**	**1**	**1**	**1**	0.969
UWaveGestureLibraryX	0.788	0.801	0.8	0.801	**0.81**	0.76
UWaveGestureLibraryY	0.728	0.714	0.721	0.711	**0.741**	0.657
UWaveGestureLibraryZ	0.749	0.752	0.752	0.74	**0.766**	0.697
UWaveGestureLibraryAll	0.91	0.889	0.899	0.891	**0.927**	0.807
Wafer	0.994	0.994	0.994	0.994	**0.996**	0.992
Wine	0.815	0.759	0.722	**0.907**	0.815	0.759
WordSynonyms	0.654	0.654	0.687	0.663	**0.691**	0.567
Worms	0.584	0.701	0.649	0.623	**0.74**	0.597
WormsTwoClass	0.597	0.688	**0.831**	0.753	0.74	0.623
Yoga	0.836	0.853	0.848	0.851	**0.87**	0.788

Table S2: Accuracy scores of the combined version of our method compared with those of DTW (unsupervised), ST and BOSS (supervised) and HIVE-COTE and EE (supervised ensemble methods), on the first 85 UCR datasets (results on the full archive were not available for comparisons). Bold scores indicate the best performing method.

Dataset	Ours (unsupervised)	Unsup.	Supervised		Supervised & ensemble	
	Combined	DTW	ST	BOSS	HIVE-COTE	EE
Adiac	0.775	0.604	0.783	0.765	**0.811**	0.665
ArrowHead	0.851	0.703	0.737	0.834	**0.863**	0.811
Beef	0.633	0.633	0.9	0.8	**0.933**	0.633
BeetleFly	0.85	0.7	0.9	0.9	**0.95**	0.75
BirdChicken	0.9	0.75	0.8	**0.95**	0.85	0.8
Car	0.817	0.733	**0.917**	0.833	0.867	0.833
CBF	0.998	0.997	0.974	0.998	**0.999**	0.998
ChlorineConcentration	**0.75**	0.648	0.7	0.661	0.712	0.656
CinCECGtorso	0.726	0.651	0.954	0.887	**0.996**	0.942
Coffee	**1**	**1**	0.964	**1**	**1**	**1**
Computers	0.704	0.7	0.736	**0.756**	0.76	0.708
CricketX	0.79	0.754	0.772	0.736	**0.823**	0.813
CricketY	0.726	0.744	0.779	0.754	**0.849**	0.805
CricketZ	0.767	0.754	0.787	0.746	**0.831**	0.782
DiatomSizeReduction	**0.99**	0.967	0.925	0.931	0.941	0.944
DistalPhalanxOutlineCorrect	0.75	0.717	**0.775**	0.728	0.772	0.728
DistalPhalanxOutlineAgeGroup	0.748	**0.77**	**0.77**	0.748	0.763	0.691
DistalPhalanxTW	0.676	0.59	0.662	0.676	**0.683**	0.647
Earthquakes	**0.748**	0.719	0.741	**0.748**	**0.748**	0.741
ECG200	0.87	0.77	0.83	0.87	0.85	**0.88**
ECG5000	0.94	0.924	0.944	0.941	**0.946**	0.939
ECGFiveDays	**1**	0.768	0.984	**1**	**1**	0.82
ElectricDevices	0.73	0.602	0.747	**0.799**	0.77	0.663
FaceAll	0.786	0.808	0.779	0.782	0.803	**0.849**
FaceFour	0.875	0.83	0.852	**1**	0.955	0.909
FacesUCR	0.893	0.905	0.906	0.957	**0.963**	0.945
FiftyWords	0.778	0.69	0.705	0.705	0.809	**0.82**
Fish	0.897	0.823	**0.989**	**0.989**	**0.989**	0.966
FordA	0.932	0.555	**0.971**	0.93	0.964	0.738
FordB	0.798	0.62	0.807	0.711	**0.823**	0.662
GunPoint	0.987	0.907	**1**	**1**	**1**	0.993
Ham	0.657	0.467	**0.686**	0.667	0.667	0.571
HandOutlines	0.922	0.881	**0.932**	0.903	**0.932**	0.889
Haptics	0.506	0.377	**0.523**	0.461	0.519	0.393
Herring	0.609	0.531	0.672	0.547	**0.688**	0.578
InlineSkate	0.418	0.384	0.373	**0.516**	0.5	0.46
InsectWingbeatSound	0.603	0.355	0.627	0.523	**0.655**	0.595
ItalyPowerDemand	0.937	0.95	0.948	0.909	**0.963**	0.962
LargeKitchenAppliances	0.861	0.795	0.859	0.765	**0.864**	0.811
Lightning2	0.77	0.869	0.738	0.836	0.82	**0.885**
Lightning7	**0.808**	0.726	0.726	0.685	0.74	0.767
Mallat	0.962	0.934	**0.964**	0.938	0.962	0.94
Meat	0.917	**0.933**	0.85	0.9	**0.933**	**0.933**
MedicalImages	**0.778**	0.737	0.67	0.718	**0.778**	0.742
MiddlePhalanxOutlineCorrect	**0.838**	0.698	0.794	0.78	0.832	0.784
MiddlePhalanxOutlineAgeGroup	0.636	0.5	**0.643**	0.545	0.597	0.558
MiddlePhalanxTW	**0.578**	0.506	0.519	0.545	0.571	0.513
MoteStrain	0.859	0.835	0.897	0.879	**0.933**	0.883

Table S2: Accuracy scores of the combined version of our method compared with those of DTW (unsupervised), ST and BOSS (supervised) and HIVE-COTE and EE (supervised ensemble methods), on the first 85 UCR datasets (results on the full archive were not available for comparisons). Bold scores indicate the best performing method.

Dataset	Ours (unsupervised)	Unsup.	Supervised		Supervised & ensemble	
	Combined	DTW	ST	BOSS	HIVE-COTE	EE
NonInvasiveFatalECGThorax1	0.938	0.79	**0.95**	0.838	0.93	0.846
NonInvasiveFatalECGThorax2	0.945	0.865	**0.951**	0.901	0.945	0.913
OliveOil	**0.9**	0.833	**0.9**	0.867	**0.9**	0.867
OSULeaf	0.814	0.591	0.967	0.955	**0.979**	0.806
PhalangesOutlinesCorrect	**0.808**	0.728	0.763	0.772	0.807	0.773
Phoneme	0.272	0.228	0.321	0.265	**0.382**	0.305
Plane	**1**	**1**	**1**	**1**	**1**	**1**
ProximalPhalanxOutlineCorrect	**0.89**	0.784	0.883	0.849	0.88	0.808
ProximalPhalanxOutlineAgeGroup	0.839	0.805	0.844	0.834	**0.859**	0.805
ProximalPhalanxTW	0.81	0.761	0.805	0.8	**0.815**	0.766
RefrigerationDevices	0.515	0.464	**0.581**	0.499	0.557	0.437
ScreenType	0.491	0.397	0.52	0.464	**0.589**	0.445
ShapeletSim	0.75	0.65	0.956	**1**	**1**	0.817
ShapesAll	0.865	0.768	0.842	**0.908**	0.905	0.867
SmallKitchenAppliances	0.717	0.643	0.792	0.725	**0.853**	0.696
SonyAIBORobotSurface1	**0.895**	0.725	0.844	0.632	0.765	0.704
SonyAIBORobotSurface2	**0.94**	0.831	0.934	0.859	0.928	0.878
StarlightCurves	0.964	0.907	0.979	0.978	**0.982**	0.926
Strawberry	0.957	0.941	0.962	**0.976**	0.97	0.946
SwedishLeaf	0.939	0.792	0.928	0.922	**0.954**	0.915
Symbols	0.954	0.95	0.882	0.967	**0.974**	0.96
SyntheticControl	0.987	0.993	0.983	0.967	**0.997**	0.99
ToeSegmentation1	0.947	0.772	0.965	0.939	**0.982**	0.829
ToeSegmentation2	0.908	0.838	0.908	**0.962**	0.954	0.892
Trace	**1**	**1**	**1**	**1**	**1**	0.99
TwoLeadECG	0.989	0.905	**0.997**	0.981	0.996	0.971
TwoPatterns	**1**	**1**	0.955	0.993	**1**	**1**
UWaveGestureLibraryX	0.81	0.728	0.803	0.762	**0.84**	0.805
UWaveGestureLibraryY	0.741	0.634	0.73	0.685	**0.765**	0.726
UWaveGestureLibraryZ	0.766	0.658	0.748	0.695	**0.783**	0.724
UWaveGestureLibraryAll	0.927	0.892	0.942	0.939	**0.968**	**0.968**
Wafer	0.996	0.98	**1**	0.995	0.999	0.997
Wine	**0.815**	0.574	0.796	0.741	0.778	0.574
WordSynonyms	0.691	0.649	0.571	0.638	0.738	**0.779**
Worms	**0.74**	0.584	**0.74**	0.558	0.558	0.662
WormsTwoClass	0.74	0.623	**0.831**	**0.831**	0.779	0.688
Yoga	0.87	0.837	0.818	**0.918**	**0.918**	0.879

Table S3: Accuracy scores of the combined version of our method compared with those of FCN (supervised), TimeNet and RWS (unsupervised), when available. Bold scores indicate the best performing method. 'X's indicate that a score were reported in the original paper, but was either obtained using transferability or on a reversed train / test split of the dataset, thus not comparable to other results for this dataset.

Dataset	Ours (unsupervised)	Supervised	Unsupervised	
	Combined	FCN	TimeNet	RWS
Adiac	0.775	**0.857**	0.565	-
ArrowHead	**0.851**	-	-	-
Beef	0.633	**0.75**	-	0.733
BeetleFly	**0.85**	-	-	-
BirdChicken	**0.9**	-	-	-
Car	**0.817**	-	-	-
CBF	0.998	**1**	-	-
ChlorineConcentration	0.75	**0.843**	0.723	0.572
CinCECGtorso	0.726	**0.813**	-	-
Coffee	**1**	**1**	-	-
Computers	**0.704**	-	-	-
CricketX	0.79	**0.815**	0.659	-
CricketY	0.726	**0.792**	X	-
CricketZ	0.767	**0.813**	X	-
DiatomSizeReduction	**0.99**	0.93	-	-
DistalPhalanxOutlineCorrect	**0.75**	-	X	-
DistalPhalanxOutlineAgeGroup	**0.748**	-	X	-
DistalPhalanxTW	**0.676**	-	X	X
Earthquakes	**0.748**	-	-	-
ECG200	**0.87**	-	-	-
ECG5000	**0.94**	-	0.934	0.933
ECGFiveDays	**1**	0.985	X	-
ElectricDevices	**0.73**	-	0.665	-
FaceAll	0.786	**0.929**	-	-
FaceFour	0.875	**0.932**	-	-
FacesUCR	0.893	**0.948**	-	-
FiftyWords	**0.778**	0.679	-	-
Fish	0.897	**0.971**	-	-
FordA	**0.932**	-	X	-
FordB	**0.798**	-	X	X
GunPoint	0.989	**1**	-	-
Ham	**0.657**	-	-	-
HandOutlines	**0.922**	-	-	0.843
Haptics	0.506	**0.551**	-	-
Herring	**0.609**	-	-	-
InlineSkate	**0.418**	0.411	-	-
InsectWingbeatSound	0.603	-	-	**0.619**
ItalyPowerDemand	0.937	**0.97**	-	0.969
LargeKitchenAppliances	**0.861**	-	-	0.792
Lightning2	0.77	**0.803**	-	-
Lightning7	0.808	**0.863**	-	-
Mallat	0.962	**0.98**	-	0.937
Meat	**0.917**	-	-	-
MedicalImages	0.778	**0.792**	0.753	-
MiddlePhalanxOutlineCorrect	**0.838**	-	X	X
MiddlePhalanxOutlineAgeGroup	**0.636**	-	X	-
MiddlePhalanxTW	**0.578**	-	X	-

Table S3: Accuracy scores of the combined version of our method compared with those of FCN (supervised), TimeNet and RWS (unsupervised), when available. Bold scores indicate the best performing method. 'X's indicate that a score were reported in the original paper, but was either obtained using transferability or on a reversed train / test split of the dataset, thus not comparable to other results for this dataset.

Dataset	Ours (unsupervised)	Supervised	Unsupervised	
	Combined	FCN	TimeNet	RWS
MoteStrain	0.859	**0.95**	-	-
NonInvasiveFatalECGThorax1	0.938	**0.961**	-	0.907
NonInvasiveFatalECGThorax2	0.945	**0.955**	-	-
OliveOil	**0.9**	0.833	-	-
OSULeaf	0.814	**0.988**	-	-
PhalangesOutlinesCorrect	**0.808**	-	0.772	-
Phoneme	**0.272**	-	-	-
Plane	**1**	-	-	-
ProximalPhalanxOutlineCorrect	**0.89**	-	X	0.711
ProximalPhalanxOutlineAgeGroup	**0.839**	-	X	X
ProximalPhalanxTW	**0.81**	-	X	-
RefrigerationDevices	**0.515**	-	-	-
ScreenType	**0.491**	-	-	-
ShapeletSim	**0.75**	-	-	-
ShapesAll	**0.865**	-	-	-
SmallKitchenAppliances	**0.717**	-	-	-
SonyAIBORobotSurface1	0.895	**0.968**	-	-
SonyAIBORobotSurface2	0.94	**0.962**	-	-
StarlightCurves	0.964	**0.967**	-	-
Strawberry	**0.957**	-	0.93	-
SwedishLeaf	0.936	**0.966**	0.901	-
Symbols	0.954	**0.962**	-	-
SyntheticControl	0.987	**0.99**	0.983	-
ToeSegmentation1	**0.947**	-	-	-
ToeSegmentation2	**0.908**	-	-	-
Trace	**1**	**1**	-	-
TwoLeadECG	0.989	**1**	-	-
TwoPatterns	**1**	0.897	0.999	0.999
UWaveGestureLibraryX	**0.81**	0.754	-	-
UWaveGestureLibraryY	**0.741**	0.725	-	-
UWaveGestureLibraryZ	**0.766**	0.729	-	-
UWaveGestureLibraryAll	**0.927**	-	-	-
Wafer	0.996	**0.997**	0.994	0.993
Wine	**0.815**	-	-	-
WordSynonyms	**0.691**	0.58	-	-
Worms	**0.74**	-	-	-
WormsTwoClass	**0.74**	-	-	-
Yoga	**0.87**	0.845	0.866	-

Table S4. Accuracy scores of variants of our method and of DTW on the remaining 43 UCR datasets, except DodgerLoopDay, DodgerLoopGame and DodgerLoopWeekend which contain missing values. Bold scores indicate the best performing method.

Dataset	Ours (unsupervised)					
	$K = 1$	$K = 2$	$K = 5$	$K = 10$	Combined	FordA ($K = 5$)
ACSF1	0.79	0.84	0.86	0.8	**0.87**	0.68
AllGestureWiimoteX	0.691	0.689	0.686	0.723	**0.733**	0.679
AllGestureWiimoteY	0.75	0.759	0.74	0.754	**0.77**	0.71
AllGestureWiimoteZ	0.701	0.704	0.713	0.711	**0.734**	0.657
BME	0.98	0.967	0.967	**0.993**	0.987	0.98
Chinatown	0.974	0.959	0.962	0.951	0.959	**0.98**
Crop	0.732	0.721	0.719	0.73	**0.747**	0.719
EOGHorizontalSignal	0.522	0.552	**0.575**	0.566	0.569	0.478
EOGVerticalSignal	0.428	0.398	0.401	0.376	0.403	0.414
EthanolLevel	0.54	**0.596**	0.584	0.482	0.588	0.318
FreezerRegularTrain	0.984	0.992	0.989	**0.998**	0.988	0.989
FreezerSmallTrain	0.925	**0.962**	0.973	0.927	0.957	0.921
Fungi	**1**	0.995	0.995	0.989	0.995	0.984
GestureMidAirD1	0.615	0.654	**0.677**	0.654	0.646	0.546
GestureMidAirD2	0.531	0.585	0.523	0.508	0.562	0.492
GestureMidAirD3	**0.323**	**0.323**	0.285	0.3	0.315	0.223
GesturePebbleZ1	0.791	**0.866**	0.86	0.802	0.82	0.494
GesturePebbleZ2	0.867	0.81	0.854	**0.905**	0.88	0.525
GunPointAgeSpan	0.991	0.991	0.984	**0.997**	0.991	0.981
GunPointMaleVersusFemale	0.994	0.994	**0.997**	**0.997**	0.994	**0.997**
GunPointOldVersusYoung	**1**	0.902	**1**	**1**	**1**	**1**
HouseTwenty	0.924	0.958	0.95	0.932	**0.966**	0.95
InsectEPGRegularTrain	**1**	**1**	**1**	**1**	**1**	**1**
InsectEPGSmallTrain	**1**	**1**	**1**	**1**	**1**	**1**
MelbournePedestrian	0.944	0.94	0.935	0.943	**0.953**	0.922
MixedShapesRegularTrain	0.906	0.909	0.92	0.905	**0.921**	0.88
MixedShapesSmallTrain	0.849	0.824	0.852	**0.866**	0.865	0.808
PickupGestureWiimoteZ	0.84	0.84	0.76	0.78	**0.86**	0.76
PigAirwayPressure	0.529	0.606	0.63	**0.659**	0.644	0.135
PigArtPressure	0.928	**0.938**	0.928	0.918	**0.938**	0.361
PigCVP	0.688	0.769	0.846	**0.889**	0.856	0.37
PLAID	0.518	**0.574**	0.567	0.54	0.542	0.447
PowerCons	**0.983**	0.944	0.939	0.928	0.944	0.922
Rock	**0.68**	0.6	0.62	0.6	0.64	0.42
SemgHandGenderCh2	0.848	**0.903**	0.862	0.831	0.885	0.802
SemgHandMovementCh2	0.729	0.747	0.769	0.804	**0.807**	0.467
SemgHandSubjectCh2	0.822	0.867	0.858	0.878	**0.904**	0.567
ShakeGestureWiimoteZ	**0.92**	0.9	0.88	0.9	0.9	0.9

S4. Multivariate Time Series

Full results corresponding to the UEA archive datasets for our method as well as the ones of DTW_D as reported by Bagnall et al. (2018) are presented in Table S5, for the unique train / test split provided in the archive.

Table S5. Accuracy scores of variants of our method on all UEA datasets, compared to DTW_D. Bold scores indicate the best performing method.

Dataset	Ours				
	$K = 5$	$K = 10$	$K = 20$	Combined	DTW_D
ArticularyWordRecognition	0.96	0.97	0.96	0.973	**0.987**
AtrialFibrillation	0.067	0.067	0.067	0.067	**0.2**
BasicMotions	**1**	0.975	**1**	**1**	0.975
CharacterTrajectories	**0.992**	0.99	0.99	**0.992**	0.989
Cricket	0.972	0.972	0.958	0.986	**1**
DuckDuckGeese	0.525	0.575	0.55	**0.625**	0.6
EigenWorms	0.802	0.817	**0.824**	0.809	0.618
Epilepsy	**0.971**	**0.971**	0.964	**0.971**	0.964
Ering	**0.133**	**0.133**	**0.133**	**0.133**	**0.133**
EthanolConcentration	0.274	0.247	0.243	0.236	**0.323**
FaceDetection	0.505	0.522	0.516	0.526	**0.529**
FingerMovements	0.54	0.55	0.52	**0.57**	0.53
HandMovementDirection	0.27	**0.338**	0.248	0.27	0.231
Handwriting	0.435	0.448	0.452	**0.466**	0.286
Heartbeat	0.688	**0.732**	0.727	0.722	0.717
InsectWingbeat	0.152	0.165	**0.168**	0.167	-
JapaneseVowels	0.984	0.981	0.984	**0.989**	0.949
Libras	0.839	0.839	0.839	0.839	**0.87**
LSST	0.553	0.564	0.571	**0.581**	0.551
MotorImagery	0.49	0.52	**0.54**	0.5	0.5
NATOPS	0.889	0.9	0.9	**0.917**	0.883
PEMS-SF	0.688	0.659	**0.711**	0.676	**0.711**
PenDigits	**0.985**	0.983	0.981	0.984	0.977
Phoneme	0.216	0.219	0.218	**0.224**	0.151
RacketSports	0.803	0.803	0.809	**0.836**	0.803
SelfRegulationSCP1	0.799	0.836	0.802	**0.819**	0.775
SelfRegulationSCP2	0.578	0.578	**0.611**	0.589	0.539
SpokenArabicDigits	0.891	0.92	0.896	0.936	**0.963**
StandWalkJump	0.333	0.333	0.267	**0.4**	0.2
UWaveGestureLibrary	0.881	0.913	0.903	**0.916**	0.903