Learnable Dynamic Temporal Pooling for Time Series Classification

Dongha Lee[1], **Seonghyeon Lee**[2], **Hwanjo Yu**[2*]
[1]Institute of Artificial Intelligence, POSTECH, Republic of Korea
[2]Dept. of Computer Science and Engineering, POSTECH, Republic of Korea
{dongha.lee, sh0416, hwanjoyu}@postech.ac.kr

Abstract

With the increase of available time series data, predicting their class labels has been one of the most important challenges in a wide range of disciplines. Recent studies on time series classification show that convolutional neural networks (CNN) achieved the state-of-the-art performance as a single classifier. In this work, pointing out that the global pooling layer that is usually adopted by existing CNN classifiers discards the temporal information of high-level features, we present a dynamic temporal pooling (DTP) technique that reduces the temporal size of hidden representations by aggregating the features at the segment-level. For the partition of a whole series into multiple segments, we utilize dynamic time warping (DTW) to align each time point in a temporal order with the prototypical features of the segments, which can be optimized simultaneously with the network parameters of CNN classifiers. The DTP layer combined with a fully-connected layer helps to extract further discriminative features considering their temporal position within an input time series. Extensive experiments on both univariate and multivariate time series datasets show that our proposed pooling significantly improves the classification performance.

Introduction

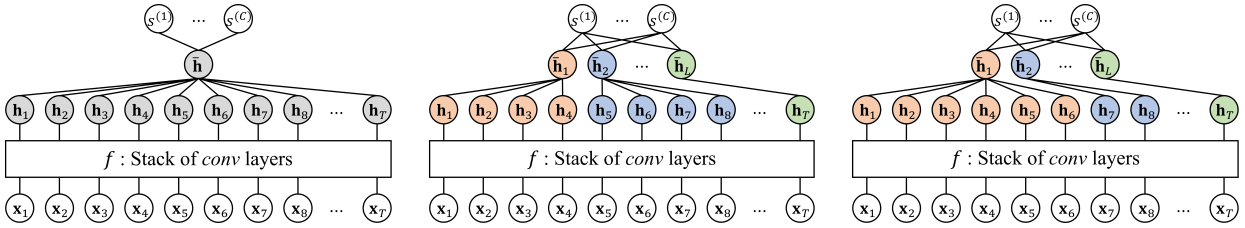
In the last two decades, time series classification has been dominated by nearest neighbor classifiers which utilize handcrafted feature-based representations (Baydogan, Runger, and Tuv 2013; Schäfer 2015) or various distance measures between time series (Zhao and Itti 2018; Yuan et al. 2019b). Recently, there have been several attempts to exploit deep neural networks (DNN) for time series classifiers (Zheng et al. 2016; Zhao et al. 2017; Wang, Yan, and Oates 2017); they do not require heavy crafting on feature engineering or data preprocessing, and easily be applied to multivariate time series as well. In practice, fully convolutional networks (FCN) and residual networks (ResNet) designed for time series classification (Wang, Yan, and Oates 2017) showed the state-of-the-art accuracy among various DNN competitors (Fawaz et al. 2019).

However, the existing convolutional neural networks (CNN) are not able to fully utilize the temporal information of high-level features for their classification. On top of convolutional layers, the CNN classifiers adopt global average pooling (GAP) or global max pooling (GMP) that simply aggregates all hidden vectors along the time axis. By doing so, they can obtain a global representation for an input time series, as well as avoid the overfitting problem with the help of much fewer model parameters. Nevertheless, such global aggregation discards the temporal position of the hidden features, which makes the CNN learn only position-invariant temporal features. Since each temporal position itself could be a useful feature for discrimination among different classes in time series classification, the global pooling layer eventually degrades the performance of the classifiers.

To address this limitation, we propose a novel pooling method that effectively reduces the temporal size (i.e., length) of network outputs while minimizing the loss of temporal information. Motivated by the observations that time series instances consist of multiple segments with distinct patterns, our dynamic temporal pooling (DTP) outputs a pooled vector for each segment rather than the one for a whole series. The DTP layer produces segment-level representations by aggregating hidden vectors in each segment, thus it enables to model the classification score based on segment-specific class weights. In other words, our CNN classifiers replace the global pooling with the segment-level pooling (being followed by a fully-connected layer), which allows extracting further class-discriminative features and improves the classification accuracy. We additionally present the class activation map (CAM) specifically for our DTP layer, indicating how much each temporal region contributes to predicting the class label of an input time series.

The challenge here is to find out consistent segments from input time series instances that are not temporally aligned with each other. To this end, the DTP layer performs semantic segmentation by using dynamic time warping (DTW). We first introduce trainable latent vectors as many as the number of segments to be identified, termed as *prototypical hidden series*, for encoding the prototypical features of each segment into them in a temporal order. Then, the DTP layer aligns the network outputs (i.e., the series of hidden vectors) with the prototypical hidden series while keeping their temporal order based on DTW; this generates the set of consecutive time points matching with each segment. In the end, we simultaneously optimize the network parameters of

*Corresponding author



(a) Global temporal pooling (b) Static temporal pooling (c) Dynamic temporal pooling

Figure 1: The architectures of CNN classifiers with different temporal pooling layers. The global pooling simply aggregates all the hidden vectors, while the proposed temporal pooling effectively reduces the temporal size based on time series segmentation.

CNN classifiers and the prototypical hidden series, thereby both of them collaboratively improve with each other. That is, training the CNN assists to capture the prototypical features of the segments, and also learning the prototypical hidden series helps the CNN to extract discriminative features.

Our empirical evaluation on extensive univariate and multivariate time series datasets demonstrates that the proposed pooling technique significantly improves the discrimination power of the CNN classifiers, regardless of its network architecture or pooling operation. The CNN classifiers with the DTP layer also beat nearest neighbor classifiers that use various distance measures by a large margin. Furthermore, we qualitatively show that the DTP layer is capable of providing an interpretable analysis on its classification result by localizing discriminative regions within a target time series.

Related Work

Deep Learning for Time Series Classification

With the great success of deep learning, a variety of deep neural networks (DNN) have been applied to time series classification, and they achieved higher accuracy than conventional nearest neighbor classifiers (Fawaz et al. 2019). Among them, convolutional neural networks (CNN) have gained much attention, such as fully convolutional networks (FCN) and residual networks (ResNet) (Wang, Yan, and Oates 2017), because of their capability of capturing local patterns as well as efficient inference by parallel computations. As presented in Figure 1, the stack of multiple convolutional layers outputs the hidden vector at each time point,[1] which eventually encodes *high-level* features about the local temporal context[2] surrounding the time point. Based on global average pooling (GAP) or global max pooling (GMP), all the hidden vectors are summarized along the time axis into a single vector (Figure 1a), and it is finally used for computing classification scores.

However, such global pooling causes the loss of information about temporal dynamics of the high-level features, and this leads to the limited performance. In case of time series classification, local temporal patterns can have different meanings depending on their temporal positions where they occur (i.e., position-variant), unlike image classification where the position of visual semantic features does not much affect its class label (i.e., position-invariant). Nevertheless, using the fully-connected layer (with local pooling after each convolution) instead of the global pooling layer, e.g., Time-LeNet (Le Guennec, Malinowski, and Tavenard 2016), not only requires a large number of parameters increasing with the length of time series but also makes the network overfitted to the training data (Fawaz et al. 2019).

Differentiable Dynamic Time Warping

Dynamic time warping (DTW) is a popular technique for measuring the distance between two time series of different lengths, based on point-to-point matching with the temporal consistency. Given two times series X and Y of length M and N, the (m, n)-th entry of its cost matrix $\Delta(X, Y) \in \mathbb{R}^{M \times N}$ represents the distance (or alignment cost) between X_m and Y_n. The DTW distance between X and Y is defined by the minimum inner product of the cost matrix and any binary alignment matrix A,

$$\mathrm{DTW}(X, Y) = \min \left\{ \langle A, \Delta(X, Y) \rangle, \forall A \in \mathcal{A} \right\}, \qquad (1)$$

where $\mathcal{A} \subset \{0, 1\}^{M \times N}$ is the set of possible binary alignment matrices whose (m, n)-th entry indicates whether X_m and Y_n are aligned or not. Each alignment matrix corresponds to a warping path that connects the upper-left $(1, 1)$-th entry to the lower-right (L, T)-th entry using $\downarrow, \rightarrow, \searrow$ moves. That is, DTW searches for the optimal warping path that minimizes the total alignment cost, and the path eventually represents the best temporal alignment between the two series. The DTW distance and its alignment matrix can be efficiently obtained by dynamic programming based on Bellman recursion, which takes a quadratic $O(MN)$ cost.

Recently, the continuous relaxation of DTW (Cuturi and Blondel 2017), named as soft-DTW, has been proposed in order to calculate the gradient of DTW with respect to its input series. Instead of the discontinuous (i.e., non-differentiable) hard-min operation taking only the minimum value in Equation (1), soft-DTW utilizes the soft-min operation with a smoothing parameter γ by adopting the concept of global alignment kernels (Cuturi et al. 2007). The soft-DTW distance between X and Y is calculated by

$$\mathrm{DTW}_\gamma(X, Y) = \min_\gamma \left\{ \langle A, \Delta(X, Y) \rangle, \forall A \in \mathcal{A} \right\},$$
$$\min\nolimits_\gamma \{a_1, \ldots, a_n\} = \begin{cases} \min_{i \le n} a_i, & \gamma = 0 \\ -\gamma \log \sum_{i=1}^{n} e^{-a_i/\gamma}, & \gamma > 0. \end{cases} \qquad (2)$$

[1] The CNN architectures, we focus on in this work, do not use any local pooling layers, so they keep the temporal size (length) of hidden representations unchanged throughout the convolutions.

[2] In a univariate case, the local contexts correspond to shapelets (Lines et al. 2012; Ma et al. 2020) of the receptive field size.

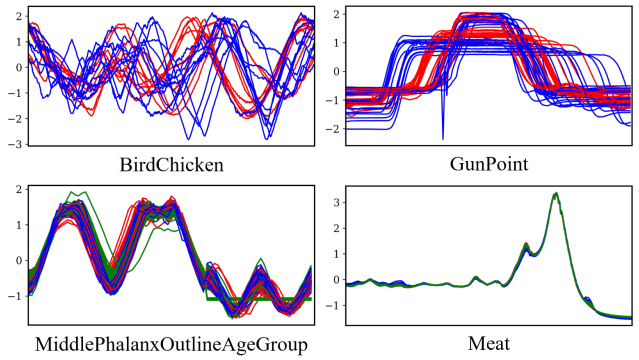


Figure 2: Examples of univariate time series, whose classes are marked in different colors. (Best viewed in color.)

Note that the original DTW is the special case of soft-DTW with $\gamma = 0$. The differentiability of soft-DTW facilitates the optimization of distance (or similarity) among time series in deep learning frameworks.

Dynamic Temporal Pooling

Problem Formulation

Given a training set of N time series instances from C classes, $\{(\mathbf{X}^1, y^1), \ldots, (\mathbf{X}^N, y^N)\}$ where $y \in \{1, \ldots, C\}$, we aim to train a CNN classifier that accurately predicts the class label of an input time series instance. For each instance $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T] \in \mathbb{R}^{D \times T}$ of length T with D variables, the network f parameterized by $\mathcal{W}$ outputs the series of hidden vectors $\mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_T] \in \mathbb{R}^{K \times T}$ of dimension size K.[3] In the end, the final classification scores $\mathbf{s} = [s^{(1)}, \ldots, s^{(C)}]$ are computed from the hidden vectors.

Temporal Pooling based on Segmentation

The purpose of our temporal pooling is to reduce the temporal size T of the hidden representation (i.e., the output of f), while minimizing the loss of temporal information in the time series. The key idea is to partition a series of hidden vectors into L segments ($L \ll T$) then generate pooled representations by summarizing the vectors in each segment. Formally, the temporal pooling layer outputs the series of pooled vectors $\bar{\mathbf{H}} = [\bar{\mathbf{h}}_1, \ldots, \bar{\mathbf{h}}_L] \in \mathbb{R}^{K \times L}$ of length L, whose l-th vector is obtained as follows.

$$\bar{\mathbf{h}}_l = \phi\left(\mathbf{h}_{t_{l-1}+1}, \mathbf{h}_{t_{l-1}+2} \ldots, \mathbf{h}_{t_l}\right), \tag{3}$$

where ϕ is the pooling operation, and $\mathcal{T}_l = \{t_{l-1}+1, \ldots, t_l\}$ is the set of consecutive time points belonging to the l-th segment ($t_0 = 0, t_L = T$). Three functions can be used for the pooling operation: calculating the average value (denoted by *avg*), the summation value (denoted by *sum*), and the maximum value (denoted by *max*) for each latent dimension.

A straightforward strategy for segmentation is to partition a whole time series into shorter ones of the same length in a static manner (Figure 1b), but it has several limitations that have to be addressed. First, time points from different time series instances are not temporally aligned in general, which makes it difficult to find absolute temporal locations for segmentation (Figure 2, upper). Furthermore, considering a segmentation task aims to discover distinct temporal patterns which are internally homogeneous, the length of each *optimal* time series segment cannot be the same as the others' in most cases (Figure 2, lower).

[3]For notational convenience, we omit the superscript n that represents the instance index in the rest of the paper.

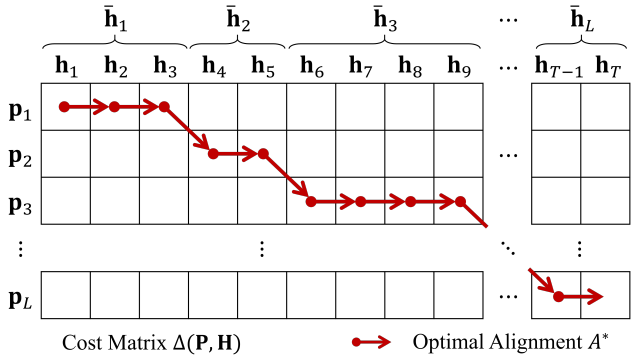


Figure 3: DTP finds L segments from the series of hidden vectors of length T based on DTW alignment, then summarizes the vectors within each segment.

To tackle these challenges, we perform semantic segmentation by matching each time point with its semantically-closest segment in a temporal order. To this end, we first introduce a *prototypical hidden series* $\mathbf{P} = [\mathbf{p}_1, \ldots, \mathbf{p}_L] \in \mathbb{R}^{K \times L}$ of length L which best summarizes the high-level features of L segments. Our proposed method, termed as dynamic temporal pooling (DTP), temporally aligns the series of prototypical hidden vectors (i.e., $\mathbf{P}$) with that of target hidden vectors (i.e., $\mathbf{H}$) by using DTW. Based on the result of DTW alignment, the series of hidden vectors is partitioned into L segments (Figure 3), and each of them is pooled by Equation (3). In this case, the optimal alignment matrix A^* between $\mathbf{P}$ and $\mathbf{H}$ can be obtained by

$$A^* = \underset{A \in \mathcal{A}}{\operatorname{argmin}} \langle A, \Delta(\mathbf{P}, \mathbf{H}) \rangle. \tag{4}$$

$\Delta \in \mathbb{R}^{L \times T}$ is the alignment cost matrix whose (l, t)-th entry $\delta(\mathbf{p}_l, \mathbf{h}_t)$ encodes the distance between $\mathbf{p}_l$ and $\mathbf{h}_t$. We define the cost by using their cosine similarity as follows.[4]

$$\delta(\mathbf{p}_l, \mathbf{h}_t) = 1 - \frac{\mathbf{p}_l \cdot \mathbf{h}_t}{\|\mathbf{p}_l\|_2 \|\mathbf{h}_t\|_2}. \tag{5}$$

Unlike eligible warping paths of conventional DTW, we need to impose an additional constraint that each time point should be aligned only with a single segment. Thus, we limit $\mathcal{A}$ to the set of possible binary alignment matrices representing a path that connects the upper-left $(1, 1)$-th entry to the lower-right (L, T)-th entry using only $\rightarrow, \searrow$ moves.

Learnable Dynamic Temporal Pooling Layer

Learning the prototypical hidden series For effective segmentation, the prototypical hidden series $\mathbf{P}$ should be optimized so that its l-th vector learns the latent semantic (or

[4]We also consider the Euclidean distance $\|\mathbf{p}_l - \mathbf{h}_t\|_2$ and the dot product $\exp(-\mathbf{p}_l \cdot \mathbf{h}_t)$, but we empirically found that the cosine distance shows the best performance among them.

Algorithm 1: Forward and backward recursions to compute $\mathrm{DTW}_\gamma(\mathbf{P}, \mathbf{H})$ and $\nabla_{\mathbf{P}}\mathrm{DTW}_\gamma(\mathbf{P}, \mathbf{H})$

Function *forward* $(\mathbf{P}, \mathbf{H})$**:**

▷ Fill the alignment cost matrix $R \in \mathbb{R}^{L \times T}$

$R_{0,0} = 0$, $R_{:,0} = R_{0,:} = \infty$

for $l = 1, \ldots, L$ **do**

  for $t = 1, \ldots, T$ **do**

    $R_{l,t} = \delta(\mathbf{p}_l, \mathbf{h}_t) + \min_\gamma\{R_{l-1,t-1}, R_{l,t-1}\}$

return $\mathrm{DTW}_\gamma(\mathbf{P}, \mathbf{H}) = R_{L,T}$

Function *backward* $(\mathbf{P}, \mathbf{H})$**:**

▷ Fill the soft alignment matrix $E \in \mathbb{R}^{L \times T}$

$E_{L+1,T+1} = 1$ ▷ $E_{l,t} := \partial R_{L,T} / \partial R_{l,t}$

$E_{:,T+1} = E_{L+1,:} = 0$

$R_{:,T+1} = R_{L+1,:} = -\infty$

for $l = L, \ldots, 1$ **do**

  for $t = T, \ldots, 1$ **do**

    $a = \exp \frac{1}{\gamma}(R_{l,t+1} - R_{l,t} - \delta(\mathbf{p}_l, \mathbf{h}_{t+1}))$

    $b = \exp \frac{1}{\gamma}(R_{l+1,t+1} - R_{l,t} - \delta(\mathbf{p}_{l+1}, \mathbf{h}_{t+1}))$

    $E_{l,t} = a \cdot E_{l,t+1} + b \cdot E_{l+1,t+1}$

return $\nabla_{\mathbf{P}}\mathrm{DTW}_\gamma(\mathbf{P}, \mathbf{H}) = \left(\frac{\partial \Delta(\mathbf{P}, \mathbf{H})}{\partial \mathbf{P}}\right)^T E$

prototypical features) corresponding to the segment l. Given a training set, we obtain the prototypical hidden series by minimizing its soft-DTW distance from the hidden representations of all the time series instances.

$$\mathcal{L}_{proto}(\mathbf{P}) = \frac{1}{N}\sum_{n=1}^{N} \mathrm{DTW}_\gamma(\mathbf{P}, f(\mathbf{X}^n; \mathcal{W})). \tag{6}$$

As we discussed, soft-DTW is differentiable with respect to its input, thus $\mathbf{P}$ can be easily optimized using the gradients $\nabla_{\mathbf{P}}\mathrm{DTW}_\gamma$ produced by Equation (6).

Similar to the original DTW, the soft-DTW distance (and the soft alignment matrix) can be obtained by solving a dynamic program based on Bellman recursion. Algorithm 1 describes the process of forward and backward recursions to compute the alignment cost $\mathrm{DTW}_\gamma(\mathbf{P}, \mathbf{H})$ and its gradient $\nabla_{\mathbf{P}}\mathrm{DTW}_\gamma(\mathbf{P}, \mathbf{H})$. Please refer to (Cuturi and Blondel 2017) for more details of deriving the algorithms. Note that a single time point should not be aligned with multiple consecutive segments. For this reason, our forward recursion does not allow the $\downarrow$ relation in its recurrence (i.e., $R_{l,t}$ depends on only $R_{l-1,t-1}$ and $R_{l,t-1}$), and accordingly, the backward recursion does not consider the $\uparrow$ relation (i.e., $E_{l,t}$ is obtained from $E_{l,t+1}$ and $E_{l+1,t+1}$).

Learning the parameters of the CNN classifier Instead of the class weight vector $\mathbf{w}^{(c)} \in \mathbb{R}^K$ that learns the importance of each latent dimension for class c, we introduce a class weight matrix $\mathbf{W}^{(c)} = [\mathbf{w}_1^{(c)}, \ldots, \mathbf{w}_L^{(c)}] \in \mathbb{R}^{K \times L}$ in order that the class weights are independently modeled for each segment. Using the pooled vectors and the class weight matrices, we calculate the classification score $s^{(c)} = \sum_{l=1}^{L} \bar{\mathbf{h}}_l \cdot \mathbf{w}_l^{(c)}$, and the posterior probability that an input time series instance belongs to class c is defined as follows:

$$P(y = c|\mathbf{X}) = \frac{\exp\left(\sum_{l=1}^{L} \bar{\mathbf{h}}_l \cdot \mathbf{w}_l^{(c)}\right)}{\sum_{c'=1}^{C} \exp\left(\sum_{l=1}^{L} \bar{\mathbf{h}}_l \cdot \mathbf{w}_l^{(c')}\right)}, \tag{7}$$

The network parameters $\mathcal{W}$ and class weight matrices $\{\mathbf{W}^{(c)}\}$ are optimized by the following classification loss.

$$\mathcal{L}_{class}(\mathcal{W}, \{\mathbf{W}^{(c)}\}) = -\frac{1}{N}\sum_{n=1}^{N} \log P(y = y^n|\mathbf{X}^n) \tag{8}$$

To sum up, our CNN classifier adopts the segment-level fully-connected layer for time series classification, by combining Equation (7) with the DTP layer.

Optimization All the parameters including the prototypical hidden series, the network parameters, and the class weight matrices are effectively optimized at the same time, by minimizing the corresponding losses:

$$\begin{aligned} &\mathbf{P} \leftarrow \mathbf{P} - \eta \cdot \partial\mathcal{L}_{proto}/\partial\mathbf{P}, \\ &\mathcal{W} \leftarrow \mathcal{W} - \eta \cdot \partial\mathcal{L}_{class}/\partial\mathcal{W}, \\ &\mathbf{W}^{(c)} \leftarrow \mathbf{W}^{(c)} - \eta \cdot \partial\mathcal{L}_{class}/\partial\mathbf{W}^{(c)}. \end{aligned} \tag{9}$$

Note that we consider $\mathbf{H} = f(\mathbf{X}; \mathcal{W})$ rather than $\mathbf{X}$ for DTW alignment (Equations (4) and (6)), so as to perform semantic segmentation in the latent space where the high-level features are embedded. For this reason, $\mathbf{P}$ and $\mathbf{H}$ collaboratively improve with each other during the training, and each of them respectively captures *stereotypical* macro-patterns and *discriminative* micro-ones. Training the network parameters of CNN classifiers (Equation (8)) helps to learn the prototypical high-level features of each segment, while optimizing the prototypical hidden series (Equation (6)) facilitates the CNN to learn further discriminative features by providing more consistent segments.

Class Activation Map for Temporal Pooling Layer

For further explanation on how much each temporal region (or time point) contributes to the classification of a target time series, we tailor a class activation map (CAM) (Zhou et al. 2016; Wang, Yan, and Oates 2017) for CNN classifiers with the DTP layer. CAM has been used to localize the regions relevant to the target class within a time series, but it cannot take into account the segment-specific class weights and also requires the class weights trained for the GAP layer.

Inspired by Grad-CAM (Selvaraju et al. 2017), we generalize the activation map for class c at time point t by

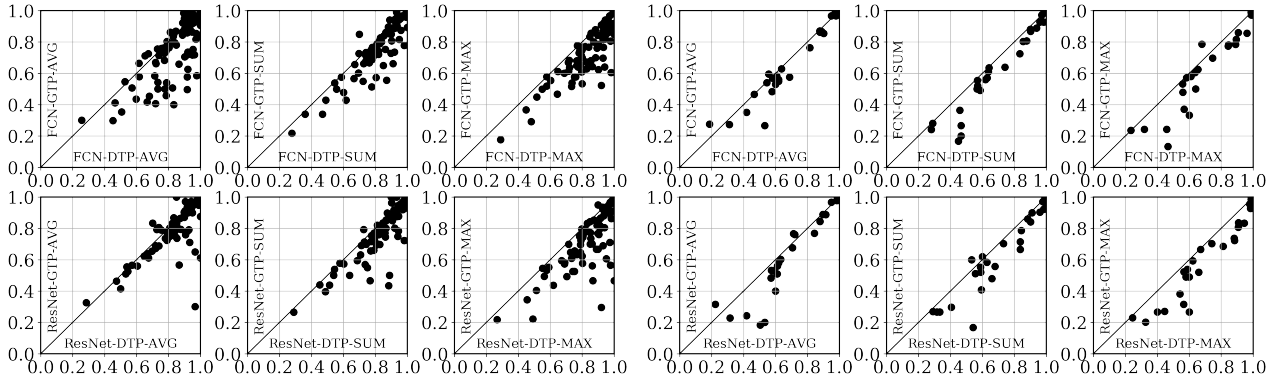
$$M_t^{(c)} = \sum_{k=1}^{K} \left(\frac{\partial s^{(c)}}{\partial h_{t,k}} \cdot h_{t,k}\right). \tag{10}$$

The point-wise gradient $\partial s^{(c)}/\partial h_{t,k}$, which implies the position-specific class weights, is used as the weight for each hidden feature $h_{t,k}$. For example, $M_t^{(c)}$ becomes equivalent to $\mathbf{w}_l^{(c)} \cdot \mathbf{h}_t$, if $\mathbf{h}_t$ is aligned with the l-th segment by the temporal sum pooling. Using the CAM, we can identify discriminative regions or temporal patterns while considering different class weights for each segment.

Experiments

Experimental Settings

Datasets For extensive evaluation, we use 85 univariate time series datasets and 30 multivariate time series datasets



(a) Datasets: UCR univariate time series archive.

(b) Datasets: UEA multivariate time series archive.

Figure 4: Comparison of the existing global temporal pooling and the proposed dynamic temporal pooling.

from the UCR/UEA repository (Bagnall et al. 2018; Dau et al. 2018). As the datasets are publicly available as well as collected from a wide range of domains, they have been widely used for the time series classification task.

Baselines As our main baselines, we use CNN classifiers equipped with different temporal pooling layers. For the experiments, we build three types of temporal pooling with avg, sum, and max operations: global temporal pooling (GTP), static temporal pooling (STP) with the fixed pooling size, and the proposed dynamic temporal pooling (DTP). We remark that several DNN architectures that have been studied for time series forecasting, for instance, dilated CNN (Oord et al. 2016), autoregressive model based on recurrent neural networks (Lai et al. 2018; Rangapuram et al. 2018), and transformer (Li et al. 2019), cannot be directly applied to time series classification (i.e., instance-level label prediction). All of them need to be customized or tuned to obtain the representation for an input time series instance; in this sense, our DTP layer can be easily generalized for the architectures as well, but we leave this for future work.

In addition, we compare our classifiers with nearest neighbor classifiers using various distance measures, especially for univariate time series.[5] Since the purpose of our work is to enhance the discrimination power of a single classifier, we exclude ensemble classifiers (Bagnall et al. 2015; Lines and Bagnall 2015; Lines, Taylor, and Bagnall 2016) that require highly intensive computations for exploiting dozens of classifiers together.

Implementation details We employ two CNN architectures, FCN and ResNet, specifically designed for time series classification (Wang, Yan, and Oates 2017); they showed the best accuracy among various types of DNNs (Fawaz et al. 2019). We implement all the CNN classifiers using PyTorch, and make use of the Numba compiler to compute the forward and backward recursions of the DTW (Algorithm 1) in parallel.[6] In order to eliminate the benefit from hyperparameter tuning, we train our classifiers without introducing the weight that balances the two losses, and also fix the number of segments L to 4 for STP and DTP.[7] Table 1 describes the details of hyperparameters used in our experiments.

Network	#Conv.	Normalize	Activate	Regularize
FCN	3	BatchNorm	ReLU	None
ResNet	9	BatchNorm	ReLU	None
Network	**Optimizer**	**Epochs**	**Batch size**	**Learn. rate**
FCN	Adam	500	16	0.0001
ResNet	Adam	500	64	0.0001

Table 1: Hyperparameters for CNN architectures and their optimization. We follow the setting provided by the previous work (Wang, Yan, and Oates 2017; Fawaz et al. 2019).

Evaluation strategy For quantitative evaluation, we conduct the pairwise posthoc analysis (Benavoli, Corani, and Mangili 2016) that statistically ranks different classifiers according to their accuracy over multiple datasets, as done in (Fawaz et al. 2019; Yuan et al. 2019b). We visualize the results by the critical difference (CD) diagram (Demšar 2006) which indicates the average rank of each classifier with thick horizontal lines showing a group of classifiers that are not significantly different ($p = 0.05$). For all the datasets, we repeatedly train each classifier three times with different random seeds, and report the median accuracy.

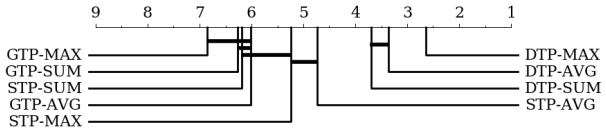
Comparison of Different Pooling Layers

We first directly compare the classification accuracy of our proposed CNN classifier (using DTP) with that of the baseline CNN classifier (using GTP). In Figure 4, a single dot represents each dataset, thus how far each dot is located from the $y = x$ line indicates the performance gap between the two pooling methods. For all the cases, we observe that most of the datasets are dotted in the lower right side of each figure, showing that DTP outperforms GTP regardless of its CNN architectures and pooling operations. In particular, the performance improvement of DTP over GTP becomes larger when it is used with the max operation. As the max pooling is effective to detect specific features in general,

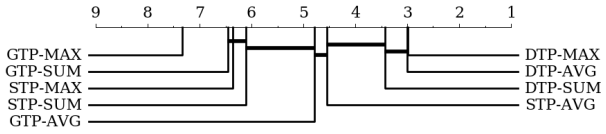
[5] This type of distance measures is not very well defined for multivariate time series.

[6] Due to the condition $L \ll T$ as well as parallel DTW computation, the training times of GTP and DTP are almost the same.

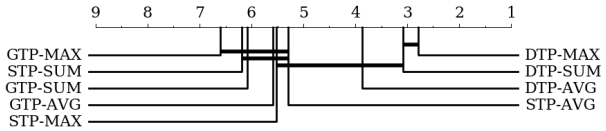
[7] The STP and DTP layers use $L = 4$ if it is not explicitly stated.



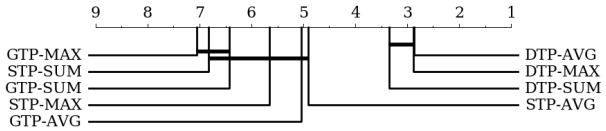
(a) Network: FCN, Datasets: UCR univariate archive.



(b) Network: ResNet, Datasets: UCR univariate archive.



(c) Network: FCN, Datasets: UEA multivariate archive.



(d) Network: ResNet, Datasets: UEA multivariate archive.

Figure 5: CD diagrams for comparing CNN classifiers that adopt different types of temporal pooling layers.

the DTP-MAX becomes good at discovering such features segment-specifically, which makes the final representation further class-discriminative.

For more statistical evaluation, we also compare different temporal pooling methods (i.e., GTP, STP, and DTP) based on pairwise statistical tests. Figure 5 presents their average rank over a bunch of datasets (85 for the univariate case, and 30 for the multivariate case) with the pairwise statistical differences. Specifically, DTP consistently performs the best among all types of temporal pooling while GTP performs the worst. In case of STP, it shows slightly better but not much statistically different performances than GTP, even though it considers the same number of segments with DTP; this implies that pooling the vectors from same-length segments at fixed temporal positions cannot effectively boost the accuracy of CNN classifiers. On the contrary, our DTP method, which uses variable-length segments identified by DTW, is capable of modeling high-level features depending on each segment, thus its discrimination power improves a lot. We can conclude that DTP successfully utilizes temporal information for its classification compared to GTP and STP.

Comparison with Nearest Neighbor Classifiers

As the nearest neighbor classifier has been one of the powerful benchmarks in the field of univariate time series classification, we compare their performances with ours. For the nearest neighbor classifier, we consider 8 different distance measures, including DTW, TWE (Marteau 2008), WDTW (Jeong, Jeong, and Omitaomu 2011), MSM (Stefan, Athitsos, and Das 2012), CID-DTW (Batista et al. 2014), shapeDTW (Zhao and Itti 2018), LWDTW (Yuan et al. 2019a), and LSDTW (Yuan et al. 2019b).

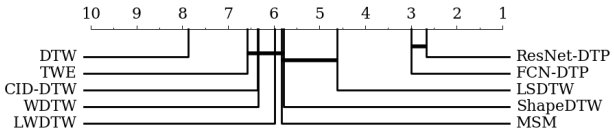


Figure 6: CD diagram for comparing CNN classifiers (DTP-MAX) with nearest neighbor classifiers.

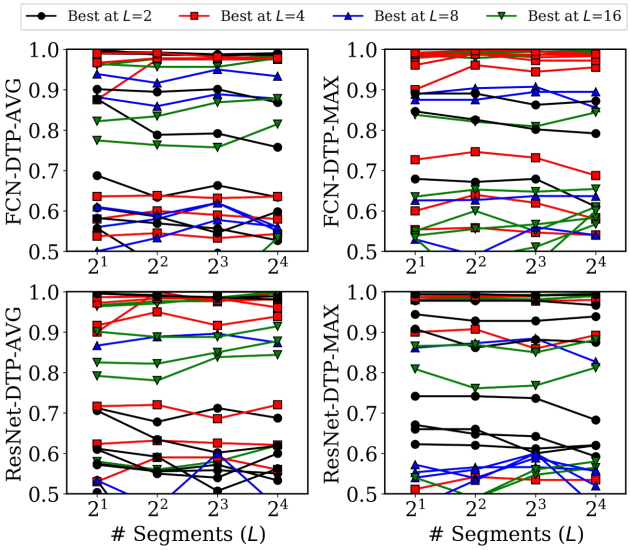
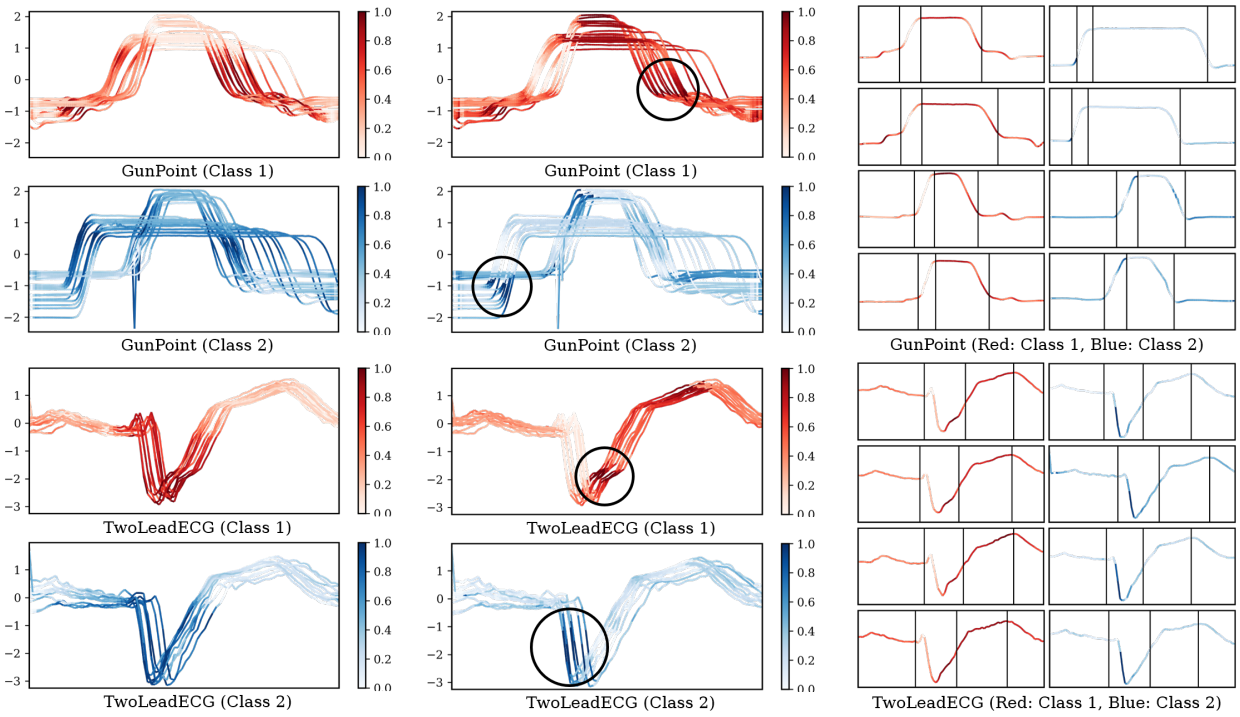


Figure 7: Performance changes of CNN classifiers with the DTP layer with respect to the number of segments L, in terms of classification accuracy.

Figure 6 clearly shows that the CNN classifiers (DTP-MAX) beat all the other classifiers by a large margin. In addition to its higher accuracy, the deep learning approach has some benefits in terms of efficiency and scalability. To predict the class label of an input time series instance, our classifiers require only a single CNN inference with an additional cost of $O(LT)$ for the DTW alignment between $\mathbf{P}$ and $\mathbf{H}$. By contrast, the nearest neighbor classifiers need to perform $O(N)$ comparisons to find the closest instance from the input time series, and each comparison takes $O(T^2)$ for computing the DTW-based distance based on point-to-point matching. This highly limits the scalability of the nearest neighbor classifiers with respect to N and T.

Parameter Analysis on L

We investigate the performance changes of the CNN classifiers (DTP-AVG and DTP-MAX) on the multivariate datasets, increasing the number of segments (i.e., L) from 2^1 to 2^4. In Figure 7, a single line denotes each dataset, and its color is determined by the optimal value of L that achieves the highest accuracy. The tendency of performance changes



(a) FCN-GTP (b) FCN-DTP (c) Segmentation results from the DTP layer

Figure 8: Time series instances from the GunPoint (Upper) and TwoLeadECG (Lower) datasets, highlighted with their class activation map. Black circles mark the discriminative temporal patterns discovered by our method. (Best viewed in color.)

is not consistent over most of the datasets, and also the optimal number of segments varies depending on the dataset. As illustrated in Figure 2, time series instances in a dataset share their own temporal patterns and lengths, which are distinguished from the ones in other datasets. For this reason, the optimal number of segments largely depends on such properties of each dataset. This result strongly indicates that finding the optimal L value for a target dataset can further enhance the performance of the CNN classifiers in practice, compared to the case of using the fixed L in our experiments.

Qualitative Analysis

To qualitatively compare the localization performance of GTP and DTP, we visualize their CAM scores by highlighting the input time series proportionally to the scores. In Figure 8a and 8b, the time series instances of class 1 and 2 are colored in red and blue, respectively, and we use the CAM scores after normalizing them in the range of $[0, 1]$ for each time series instance. Although the two CNN classifiers achieve almost the same accuracy for both the datasets (i.e., GunPoint and TwoLeadECG), they highlight different regions as the discriminative temporal patterns that most contribute to predicting its class label. It is worth noting that the localized regions discovered by DTP are more clearly distinguishable between the classes compared to GTP, which results in better interpretability of the CNN classifiers.

Furthermore, Figure 8c provides segmentation results obtained from the DTP layer; each time series instance is divided into four segments by vertical lines. The l-th segments ($l = 1, \ldots, 4$) of all instances share similar (or consistent) temporal patterns, even where their original input series are not temporally aligned. The results support that our prototypical hidden series $\mathbf{P}$ successfully encodes the prototypical high-level features of the segments in a temporal order, thereby the DTP layer can perform semantic segmentation based on the DTW alignment between $\mathbf{P}$ and $\mathbf{H}$.

Conclusion

This paper proposes a dynamic temporal pooling, termed as DTP, which outputs the pooled vectors from temporally-ordered segments; this enables CNN classifiers to make use of the segment-level fully-connected layer for time series classification. We present a learning framework to simultaneously optimize the network parameters of a CNN classifier and the prototypical hidden series that encodes the latent semantic of the segments. By finding the optimal alignment between the prototypical hidden series and the hidden representation of a target time series, the DTP layer is able to partition the whole series into a fixed number of segments. Our extensive experiments show that the proposed DTP layer outperforms other baseline pooling layers, and it successfully identifies consistent segments from time series instances that are out of temporal alignment.

Acknowledgements

This work was supported by the NRF grant funded by the MSIT (No. 2020R1A2B5B03097210), and the IITP grant funded by the MSIT (No. 2018-0-00584, 2019-0-01906).

References

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

Bagnall, A.; Lines, J.; Hills, J.; and Bostrom, A. 2015. Time-series classification with COTE: the collective of transformation-based ensembles. *IEEE Transactions on Knowledge and Data Engineering* 27(9): 2522–2535.

Batista, G. E.; Keogh, E. J.; Tataw, O. M.; and De Souza, V. M. 2014. CID: an efficient complexity-invariant distance for time series. *Data Mining and Knowledge Discovery* 28(3): 634–669.

Baydogan, M. G.; Runger, G.; and Tuv, E. 2013. A bag-of-features framework to classify time series. *IEEE transactions on pattern analysis and machine intelligence* 35(11): 2796–2802.

Benavoli, A.; Corani, G.; and Mangili, F. 2016. Should we really use post-hoc tests based on mean-ranks? *The Journal of Machine Learning Research* 17(1): 152–161.

Cuturi, M.; and Blondel, M. 2017. Soft-DTW: a Differentiable Loss Function for Time-Series. In *International Conference on Machine Learning*, 894–903.

Cuturi, M.; Vert, J.-P.; Birkenes, O.; and Matsui, T. 2007. A kernel for time series based on global alignments. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 2, II–413. IEEE.

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.; Batista, G.; and Hexagon-ML. 2018. The UCR Time Series Classification Archive. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/, Date last accessed 02/19/2021.

Demšar, J. 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research* 7(Jan): 1–30.

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

Jeong, Y.-S.; Jeong, M. K.; and Omitaomu, O. A. 2011. Weighted dynamic time warping for time series classification. *Pattern recognition* 44(9): 2231–2240.

Lai, G.; Chang, W.-C.; Yang, Y.; and Liu, H. 2018. Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, 95–104.

Le Guennec, A.; Malinowski, S.; and Tavenard, R. 2016. Data Augmentation for Time Series Classification using Convolutional Neural Networks. In *Proceedings of AALTD 2016: Second ECML/PKDD International Workshop on Advanced Analytics and Learning on Temporal Data*, 11.

Li, S.; Jin, X.; Xuan, Y.; Zhou, X.; Chen, W.; Wang, Y.-X.; and Yan, X. 2019. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In *Advances in Neural Information Processing Systems*, 5243–5253.

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

Lines, J.; Davis, L. M.; Hills, J.; and Bagnall, A. 2012. A shapelet transform for time series classification. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 289–297.

Lines, J.; Taylor, S.; and Bagnall, A. 2016. Hive-cote: The hierarchical vote collective of transformation-based ensembles for time series classification. In *2016 IEEE 16th international conference on data mining (ICDM)*, 1041–1046. IEEE.

Ma, Q.; Zhuang, W.; Li, S.; Huang, D.; and Cottrell, G. W. 2020. Adversarial Dynamic Shapelet Networks. *Proceedings of the AAAI Conference on Artificial Intelligence* 34(04): 5069–5076.

Marteau, P.-F. 2008. Time warp edit distance with stiffness adjustment for time series matching. *IEEE transactions on pattern analysis and machine intelligence* 31(2): 306–318.

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

Rangapuram, S. S.; Seeger, M. W.; Gasthaus, J.; Stella, L.; Wang, Y.; and Januschowski, T. 2018. Deep state space models for time series forecasting. In *Advances in neural information processing systems*, 7785–7794.

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

Selvaraju, R. R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; and Batra, D. 2017. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, 618–626.

Stefan, A.; Athitsos, V.; and Das, G. 2012. The move-split-merge metric for time series. *IEEE transactions on Knowledge and Data Engineering* 25(6): 1425–1438.

Wang, Z.; Yan, W.; and Oates, T. 2017. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks*, 1578–1585. IEEE.

Yuan, J.; Douzal-Chouakria, A.; Yazdi, S. V.; and Wang, Z. 2019a. A large margin time series nearest neighbour classification under locally weighted time warps. *Knowledge and Information Systems* 59(1): 117–135.

Yuan, J.; Lin, Q.; Zhang, W.; and Wang, Z. 2019b. Locally Slope-based Dynamic Time Warping for Time Series Classification. In *Proceedings of the 28th ACM International Con-*

ference on Information and Knowledge Management, 1713–1722.

Zhao, B.; Lu, H.; Chen, S.; Liu, J.; and Wu, D. 2017. Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics* 28(1): 162–169.

Zhao, J.; and Itti, L. 2018. shapedtw: Shape dynamic time warping. *Pattern Recognition* 74: 171–184.

Zheng, Y.; Liu, Q.; Chen, E.; Ge, Y.; and Zhao, J. L. 2016. Exploiting multi-channels deep convolutional neural networks for multivariate time series classification. *Frontiers of Computer Science* 10(1): 96–112.

Zhou, B.; Khosla, A.; Lapedriza, A.; Oliva, A.; and Torralba, A. 2016. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2921–2929.