Multivariate Time Series Classification with Hierarchical Variational Graph Pooling

Haoyan Xu[1*], Ziheng Duan[1*], Yunsheng Bai[2*]
Yida Huang[1], Anni Ren[3], Qianru Yu[1], Qianru Zhang[4]
Yueyang Wang[5†], Xiaoqian Wang[6†] Yizhou Sun[2†], Wei Wang[2†],
[1]Zhejiang University
[2]University of California, Los Angeles
[3]Nanjing University of Aeronautics and Astronautics
[4]Harbin Institute of Technology
[5]Chongqing University
[6]Purdue University
haoyanxu@zju.edu.cn, duanziheng@zju.edu.cn, yba@g.ucla.edu
stevenhuang@zju.edu.cn, qianruy@zju.edu.cn, qianruy@zju.edu.cn, irenechzhang@gmail.com
yueyangw@cqu.edu.cn, joywang@purdue.edu, yzsun@cs.ucla.edu, weiwang@cs.ucla.edu

Abstract

Over the past decade, multivariate time series classification (MTSC) has received great attention with the advance of sensing techniques. Current deep learning methods for MTSC are based on convolutional and recurrent neural network, with the assumption that time series variables have the same effect to each other. Thus they cannot model the pairwise dependencies among variables explicitly. What's more, current spatial-temporal modeling methods based on GNNs are inherently flat and lack the capability of aggregating node information in a hierarchical manner. To address this limitation and attain expressive global representation of MTS, we propose a graph pooling based framework MTPool and view MTSC task as graph classification task. With graph structure learning and temporal convolution, MTS slices are converted to graphs and spatial-temporal features are extracted. Then, we propose a novel graph pooling method, which uses an "encoder-decoder" mechanism to generate adaptive centroids for cluster assignments. GNNs and graph pooling layers are used for joint graph representation learning and graph coarsening. With multiple graph pooling layers, the input graphs are hierachically coarsened to one node. Finally, differentiable classifier takes this coarsened one-node graph as input to get the final predicted class. Experiments on 10 benchmark datasets demonstrate MTPool outperforms state-of-the-art methods in MTSC tasks.

Introduction

Multivariate time series (MTS), where a sequence of measurements from multiple variables or sensors are collected, is used in various fields of studies. MTS have the following two major characteristics: (1) Each univariate time series has internal temporal dependency pattern; (2) There always exists hidden dependency relationships among different variables in MTS. Owing to the advanced sensing techniques, the Multivariate Time Series Classification (MTSC) problem, identifying the labels for MTS records, has attracted a great amount of attention in recent decades (Zhang et al. 2020). Multivariate time series classification models have been applied in many different real-world applications such as sleep stage identification (Sun et al. 2019), healthcare (Kang and Choi 2014) and action recognition (Yu and Lee 2015).

Plenty of MTS classification algorithms have been developed over the years. Distance-based methods such as Dynamic Time Warping (DTW) with k-NN (Seto, Zhang, and Zhou 2015) and feature-based methods such as Hidden Unit Logistic Model (HULM) (Pei et al. 2017) have proven to be successful in classification task on many benchmark MTS datasets. However, these approaches need heavy crafting on data preprocessing and feature engineering. Recently, many deep learning based methods are exploited for end-to-end MTS classification. Fully convolutional networks (FCN) and the residual networks (ResNet) can achieve comparable or better performance than traditional methods (Wang, Yan, and Oates 2017). MLSTM-FCN (Karim et al. 2019) uses a LSTM layer and a stacked CNN layer along with squeeze-and-excitation blocks to generate latent features. These deep learning based methods have achieve promising performance in MTSC tasks. However, all the existing deep learning based methods do not model the latent interdependency among variables of MTS and the model's input must be grid data, this limits the representation ability of these models.

Graphs are a special form of data which describe the relationships between different entities. Graph Neural Networks (GNNs) employ deep neural networks to aggregate feature information of neighboring nodes, which makes the aggregated embedding more powerful. Multivariate time series classification can be viewed naturally from a graph perspective. Variables from multivariate time series can be considered as nodes in a graph, and they are inter-linked through

[*]Equal contribution with order determined by rolling the dice.
[†]Corresponding authors.

their hidden dependency relationships. It follows that modeling multivariate time series data using graph neural networks can be a promising way to preserve their temporal trajectory while exploiting the interdependency among time series (Wu et al. 2020). Therefore, in this paper, we propose to convert MTS slices to graphs through graph structure learning and view MTS classification task as graph classification task.

The task of graph classification is to predict the type of a given graph by utilizing the input graph structure and initial node-level representations. For example, given a molecule, the task could be to predict if it is toxic (Ranjan, Sanyal, and Talukdar 2020). Current GNNs are inherently flat and lack the capability of aggregating node information in a hierarchical manner. To address this limitation, hierarchical pooling methods such as gPool (Gao and Ji 2019), DiffPool (Ying et al. 2018), MemGNN (Khasahmadi et al. 2020) have been proposed and achieve promising results in many graph classification tasks. In hierarchical pooling, sets of nodes are recursively aggregated to form a cluster that represents a node in the pooled graph. gPool downsamples by selecting the most important nodes. DiffPool downsamples by clustering the nodes using GNNs. Similar with DiffPool, MemGNN use a multi-head array of memory keys and a convolution operator to aggregate the soft cluster assignments from different heads, and compute the attention scores between nodes and clusters.

To model temporal pattern and hidden dependency relationships among variables in MTS, as well as exploiting and integrating global representation of MTS, we propose a novel MTS classification framework called MTPool (Multivariate Time Series Classification with Variational Graph Pooling). We construct adjacency matrix through graph structure learning and feature matrix through temporal convolution. GNNs are used to aggregate and fuse spatial-temporal features. Then with multiple graph pooling layers, the input graphs are hierachically coarsened to one node to get their graph-level representations. Finally, these final output graph-level embeddings can be used as features input to a differentiable classifier for MTSC tasks.

During experiments, we found out that many existing graph pooling methods can be incorporated into our MTPool framework and achieve relatively good performance. However, we notice that some of these state-of-the-art pooling mechanisms such as MemGNN (Khasahmadi et al. 2020) involve a process of generating centroids for soft cluster assignments, but the generation of centroids is not input-related to input graphs which does not make sense because the centroids for different graphs should be different. MemGNN randomly initialize centroids (keys) before training and makes them be trainable parameters. However, although centroids are learnable during training, they can not change when testing. This results in all the testing graphs can only use the same centroids as train graphs' for cluster assignments. It is evident that each input graph should has its specific centroids based on its own topology structure and node features. Thus we propose a novel pooling layer Variational Pooling. The "encoder-decoder" architecture in Variational Pooling enables the generation process of centroids is input-related and still keep the property of permutation invariance, making the model more inductive and leading to better performance. Thus our main contributions are as follows:

- To the best of our knowledge, we are the first to propose a hierarchical graph pooling based framework to model MTS and hierarchically generate its global representation for MTS classification.
- We design MTPool as an end-to-end joint framework for graph structure learning, temporal convolution, graph representation learning and graph coarsening.
- We propose a novel pooling method Variational Pooling. The centroids for cluster assignments is input-related to the input graphs, which makes the model more inductive and leads to better performance.
- We conduct extensive experiments on MTS benchmark datasets, and the empirical results prove that the performance of the proposed method is better than state-of-the-art models in most cases.

Related Work

Multivariate Time Series Classification

Most MTSC methods can be grouped into three categories: distance-based, feature-based and deep learning based methods. Here we only discuss about deep learning based methods.

Currently, two popular deep learning models, CNN and RNN, are widely used in MTS classification. These models often use a LSTM layer and stacked CNN layer to extract features from the time series, and a softmax layer is then applied to predict the label (Zhang et al. 2020). MLSTM-FCN (Karim et al. 2019) uses a LSTM layer and a stacked CNN layer along with squeeze-and-excitation blocks to generate latent features. TapNet (Zhang et al. 2020) also constructs a LSTM layer and a stacked CNN layer, followed by an attentional prototype network.

Deep learning based methods require less domain knowledge in time series data than traditional methods. However, the limitation of the above models is obvious: they assume that the time series variables have the same effect to each other, thus they cannot model the pairwise dependencies among variables explicitly. Hence, graph is the most appropriate data structure for modeling MTS.

Graph Neural Networks

The concept of graph neural network (GNN) was first proposed in (Scarselli et al. 2008), which extended existing neural networks for processing the data represented in graph domains. GNNs follow a neighborhood aggregation scheme, where the representation vector of a node is computed by recursively aggregating and transforming representation vectors of its neighboring nodes. Many GNN variants have been proposed and have achieved state-of-the-art results on both node and graph classification tasks. For example, the Graph Convolutional Networks (GCN) (Kipf and Welling 2016) could be regarded as an approximation of spectral-domain convolution of the graph signals. GraphSAGE (Hamilton, Ying, and Leskovec 2017) and FastGCN (Chen, Ma, and

Xiao 2018) sample and aggregate of the neighborhood information while enabling training in batches but sacrificing some time-efficiency. Graph Attention Networks (GAT) (Veličković et al. 2017) designs a new way to gather neighbors through self-attention. After this, Graph Isomorphism Network (GIN) (Xu et al. 2018) and k-GNNs (Morris et al. 2019) are developed, introducing more complex and diverse forms of aggregation.

Graph Pooling

Graph pooling methods can be classified to three categories: topology based, global, and hierarchical pooling. Here we only discuss about hierarchical pooling methods. DiffPool (Ying et al. 2018) trains two parallel GNNs to compute node representations and cluster assignments. gPool (Gao and Ji 2019) and SAGPool (Lee, Lee, and Kang 2019) drop nodes from the original graph rather than group multiple nodes to form a cluster in the pooled graph. They devise a top-K node selection procedure to form an induced sub-graph for the next input layer. Although they are more efficient than DiffPool, they do not aggregate nodes nor compute soft edge weights. This makes them unable to preserve node and edge information effectively. MemGNN (Khasahmadi et al. 2020) also learn soft cluster assignments, and they use a clustering-friendly distribution to compute the attention scores between nodes and clusters. However, they generate centroids (which are used to compute soft assignments) without involving the input graphs, which does not make sense intuitively because the centroids for different input graphs should be different. To address this limitation, we propose Variational Pooling, which use an "encoder-decoder" architecture to generate centroids to keep the property of permutation invariance while making centroids input-related to graphs.

Problem Formulation

A multivariate time series (MTS) can be represented as a matrix $\mathbf{X} = \{x_1, x_2, ..., x_n\} \in \mathbb{R}^{n \times T}$, in which T is the length of the time series and n is the number of multivariate dimensions. Each MTS is associated with a class label y from a predefined label set. Given a group of MTS $\boldsymbol{\mathcal{X}} = \{\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_N\} \in \mathbb{R}^{N \times n \times T}$, and the corresponding labels $\mathcal{Y} = \{y_1, y_2, ..., y_N\} \in \mathbb{R}^N$, the research goal is to learn the mapping relationship between $\boldsymbol{\mathcal{X}}$ and $\mathcal{Y}$ based on the proposed model.

Framework

In this section, we present the proposed MTPool in detail. The schematic of MTPool is shown in Fig 1.

Graph Structure Learning

We propose a dynamic relation embedding strategy, which learns the adjacency matrix $\mathbf{A}^{(i)} \in \mathbb{R}^{n \times n}$ adaptively to model latent relations in the given time series sample $\mathbf{X}_i$. The learned graph structure (adjacency matrix) $\mathbf{A}^{(i)}$ is defined as:

$$\mathbf{A}^{(i)} = Embed_1(\mathbf{X}_i) \tag{1}$$

For the input MTS $\mathbf{X}_i = \{x_1, x_2, ..., x_n\} \in \mathbb{R}^{n \times T}$ from the i_{th} sample, where x_i, x_j denote the i^{th}, j^{th} time series. We first calculate the similarity matrix $\mathbf{C}^{(i)}$ between sampled time series:

$$\mathbf{C}_{ij}^{(i)} = \frac{exp(-\sigma(distance(x_i, x_j)))}{\sum_{p=0}^{n} exp(-\sigma(distance(x_i, x_p))} \tag{2}$$

where $distance$ denotes the distance metric such as Euclidean Distance, Absolute Value Distance, Dynamic Time Warping, etc. Then the dynamic adjacency matrix $\mathbf{A}^{(i)}$ can be calculated as:

$$\mathbf{A}^{(i)} = \sigma(\mathbf{C}^{(i)}\mathbf{W}) \tag{3}$$

where $\mathbf{W}$ is learnable model parameters, σ is an activation function. What's more, to improve training efficiency, reduce noise impact and make the model more robust, threshold c_1 is set to make the adjacency matrix sparse:

$$\mathbf{A}_{ij}^{(i)} = \begin{cases} \mathbf{A}_{ij}^{(i)} & \mathbf{A}_{ij}^{(i)} >= c_1 \\ 0 & \mathbf{A}_{ij}^{(i)} < c_1 \end{cases} \tag{4}$$

Finally, normalization is applied to $\mathbf{A}^{(i)}$.

Temporal Convolution

This stage aims to extract temporal features and construct feature matrix $\mathbf{X}^{(i)}$. With temporal convolution, we can get each MTS's feature matrix as follows:

$$\mathbf{X}^{(i)} = Embed_2(\mathbf{X}_i) \in \mathbb{R}^{n \times d} \tag{5}$$

When analyzing time series, it is necessary to consider not only its numerical value but also its trend over time. Time series in the real world usually have multiple simultaneous periodicities. For example, the population of a certain city not only shows a certain trend every day, but also a meaningful pattern can be observed by observing it on the scale of one week or one month. Therefore, it is reasonable and necessary to extract the features of the time series in units of multiple specific periods. To simulate this situation, we use multiple CNN filters with different receptive fields, namely kernel sizes, to extract features at multiple time scales.

For the i-th CNN filters, given the input time series X, the feature vector h_i are extracted as follows: $h_i = \sigma(W_i * X + b)$, where $*$ denotes the convolution operation, σ is a nonlinear activation function, such as $RELU(x) = max(0, x)$, W_i represents the i-th CNN kernel and b is the bias. And the final feature vector can be expressed as $h = [h_1, h_2, ..., h_p]$, where p is the CNN filters number and $[*]$ means concatenate operation. In this way, features under different periods are extracted, which provides effective information for time series classification.

Spatial-temporal Modeling

In this stage, k GNN layers $(G_1, G_2, ...G_k)$ are employed on the input graphs (denoted as $\mathbf{X}^{(i)}, \mathbf{A}^{(i)}$) for spatial-temporal modeling. GNN layers can fuse spatial dependencies and temporal patterns for embedding feature of nodes and transform the feature dimension of nodes to d_{encode}, just as shown in equation (6):

$$\mathbf{X}_{encode}^{(i)}, \mathbf{A}_{encode}^{(i)} = G_k(G_{k-1}(...G_1(\mathbf{X}^{(i)}, \mathbf{A}^{(i)}))) \tag{6}$$

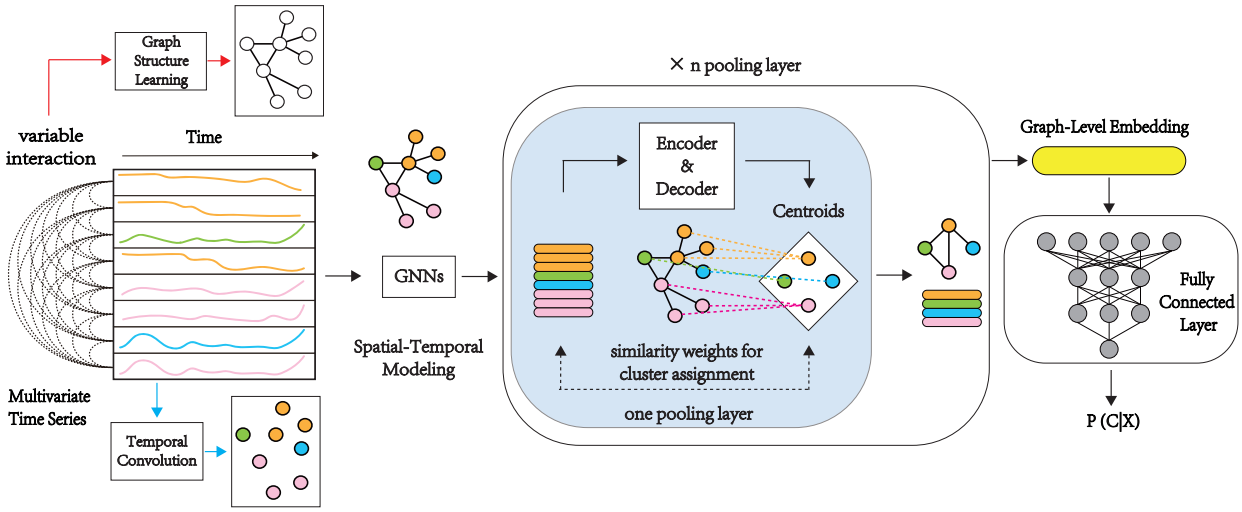

Figure 1: The general architecture of MTPool. Adjacency matrix and feature matrix are constructed through graph structure learning and temporal convolution respectively. Then GNNs aggregate and fuse spatial-temporal features. Then with multiple graph pooling layers, the input graphs are hierachically coarsened to one node to attain their graph-level representations. Finally, these final output graph-level embeddings can be used as features input to a differentiable classifier for MTSC tasks. MTPool is an end-to-end joint framework for graph structure learning, temporal convolution, graph representation learning and graph coarsening.

where $i = 1, 2, ...N$, $\mathbf{X}^{(i)}_{encode} \in \mathbb{R}^{n \times d_{encode}}$, $\mathbf{A}^{(i)}_{encode} \in \mathbb{R}^{n \times n}$. G_j $(j = 1, 2, ..., k)$ is consisted of a graph neural network layer GNN and a batch normalization layer. GNN can be such as GCN (Defferrard, Bresson, and Vandergheynst 2016), GAT (Veličković et al. 2017), GIN (Xu et al. 2018), etc.

Variational Graph Pooling

Overall Transformation In this stage, the encoded graph is hierarchically pool to a single node for generating its graph-level representation. A pooling layer can pool each encoded graph, $\mathbf{X}^{(i)}_{encode} \in \mathbb{R}^{n \times d_{encode}}$ and $\mathbf{A}^{(i)}_{encode} \in \mathbb{R}^{n \times n}$, to a specific coarsened graph, $\mathbf{X}^{(i)}_{pool} \in \mathbb{R}^{n_{pool} \times d_{pool}}$ and $\mathbf{A}^{(i)}_{pool} \in \mathbb{R}^{n_{pool} \times n_{pool}}$, which has less nodes than the input graph. The overall transformation of one pooling layer is shown in equation 7, where σ is a non-linear activation function, $\mathbf{W} \in \mathbb{R}^{d_{encode} \times d_{pool}}$ is a trainable parameter matrix standing for a linear transformation and $\mathbf{S}^{(i)} \in \mathbb{R}^{n_{pool} \times n}$ is an assignment matrix representing a projection from the original nodes to pooled nodes (clusters).

$$\begin{aligned} \mathbf{X}^{(i)}_{pool} &= \sigma\left(\mathbf{S}^{(i)}\mathbf{X}^{(i)}_{encode}\mathbf{W}\right) \\ \mathbf{A}^{(i)}_{pool} &= \sigma\left(\mathbf{S}^{(i)}\mathbf{A}^{(i)}_{encode}(\mathbf{S}^{(i)})^{\mathsf{T}}\right) \end{aligned} \tag{7}$$

After stacking several pooling layers, we can pool the original graph to a single node and get its graph-level representation vector $\mathbf{X}_{final}$ as follows:

$$\mathbf{X}^{(i)}_{final} = P_k(P_{k-1}(...P_1(\mathbf{X}^{(i)}_{encode}))) \tag{8}$$

where P_j $(j = 1, 2, ..., k)$ represents one pooling layer.

How to Compute Assignment Matrix Different approaches are adopted to compute the assignment matrix $\mathbf{S}^{(i)}$ in different pooling methods. DiffPool trains two parallel GNNs and learns a soft cluster assignment for nodes at each layer of a deep GNN. MemGNN uses a multi-head array of memory keys and a convolution operator to aggregate the soft cluster assignments from different heads. Although MemGNN uses a clustering-friendly distribution to compute the attention scores between nodes and clusters and performs better than DiffPool in many tasks, we notice that it generates memory heads, which stands for the new centroids in the space of pooled graphs, without involvement of the input graphs. However, it's obvious that centroids for different graphs should be different and each input graph should has its corresponding centroids based on its own topology structure and node features. Thus here we propose a new method: Variational Pooling, to calculate $\mathbf{S}^{(i)}$. We first generate h batches of centroids $\mathbf{K}^{(i)} \in \mathbb{R}^{h \times n_{pool} \times d_{encode}}$ based on the input graph and then compute and aggregate the relationship between every batch of centroids and the encoded graph for assignment matrix $\mathbf{S}^{(i)}$.

How to Generate Centroids The generation of centroids $\mathbf{K}^{(i)}$ should satisfy the following properties:

- **Permutation invariance.** The same graph can be represented by different adjacency matrices by permuting the order of nodes. The centroids of the same graph should be invariant to such changes.
- **Input-related.** The generation of centroids should be based on the input graph. If the input graph changes, the centroids should change accordingly to capture global fea-

Table 1: Summary of the 10 UEA datasets used in experimentation.

	Name	Train Size	Test Size	Num Series	Series Length	Classes
AF	AtrialFibrillation	15	15	2	640	3
FM	FingerMovements	316	100	28	50	2
HMD	HandMovementDirection	160	74	10	400	4
HB	Heartbeat	204	205	61	405	2
LIB	Libras	180	180	2	45	15
MI	MotorImagery	278	100	64	3000	2
NATO	NATOPS	180	180	24	51	6
PD	PenDigits	7494	3498	2	8	10
SRS2	SelfRegulationSCP2	200	180	7	1152	2
SWJ	StandWalkJump	12	15	4	2500	3

tures effectively.

- **Correct dimension.** The dimension of centroids matrix $\mathbf{K}^{(i)}$ should be adjusted according to the dimension of the input graph and the dimension of the output coarsened graph we need.

Therefore, we propose an "encoder-decoder" architecture for computing centroids matrix $\mathbf{K}^{(i)}$. In general, the *encoder* ensures the property of permutation invariance and makes the centroids adaptive to input graphs while the *decoder* controls the dimension of the output coarsened graph.

Generating Centroids As shown in equation 9, an *encoder* is deployed over the input graph, transforming $\mathbf{X}^{(i)}_{encode} \in \mathbb{R}^{n \times d_{encode}}$ to $\mathbf{X}^{(i)}_{g} \in \mathbb{R}^{1 \times d_{encode}}$. Then a *decoder* is applied to map $\mathbf{X}^{(i)}_{g}$ to $\mathbf{K}^{(i)} \in \mathbb{R}^{(h \times n_{pool}) \times d_{encode}}$, after which $\mathbf{K}^{(i)}$ is reshaped to $\mathbb{R}^{h \times n_{pool} \times d_{encode}}$.

$$\mathbf{K}^{(i)} = Decoder\left(Encoder\left(\mathbf{X}^{(i)}_{encode}\right)\right) \tag{9}$$

The expression of encoder and decoder we use in this paper is as follows:

$$\begin{aligned} Encoder: \mathbf{X}^{(i)}_{g} &= \sum_{i=1}^{n} \sigma(u_i^T x_{avg} u_i) \\ &= \sum_{i=1}^{n} \sigma_1(u_i^T \sigma_2((\frac{1}{n}\sum_{j=1}^{n} u_j)\mathbf{W})u_i) \\ Decoder: MLP \end{aligned} \tag{10}$$

where u_i is the embedding of node i, σ_1 is the sigmoid function and σ_2 is the $tanh$ activation function. For encoder, we use attention mechanism to let the model learn weights guided by the specific task to generate graph-level representation $\mathbf{X}^{(i)}_{g}$ for centroids. For decoder, here we use Multiple Layer Perceptron (MLP) to reshape matrix $\mathbf{K}^{(i)}$ to the size we need.

Computing Assignment Matrix With centroids matrix $\mathbf{K}^{(i)}$, we can compute the relationship $\mathbf{S}^{(i)}_{p} \in \mathbb{R}^{n_{pool} \times n}$ $(p = 1, 2, ..., h)$ between every batch of centroids $\mathbf{K}^{(i)}_{p} \in \mathbb{R}^{n_{pool} \times d_{encode}}$ $(p = 1, 2, ..., h)$ and $\mathbf{X}^{(i)}_{encode} \in \mathbb{R}^{n \times d_{encode}}$.

We use cosine similarity to evaluate the relationship between input node embeddings and centroids, as described in equation 11, followed by a row normalization deployed in the resulting assignment matrix.

$$\begin{aligned} \mathbf{S}^{(i)}_{p} &= \text{cosine}\left(\mathbf{K}^{(i)}_{p}, \mathbf{X}^{(i)}_{encode}\right) \\ \mathbf{S}^{(i)}_{p} &= \text{normalize}\left(\mathbf{S}^{(i)}_{p}\right) \end{aligned} \tag{11}$$

We finally aggregate the information of h relationship $\mathbf{S}^{(i)}_{p} \in \mathbb{R}^{n_{pool} \times n}$. In (12), we concatenate $\mathbf{S}^{(i)}_{p}$ $(p = 1, 2, ..., h)$ and perform a trainable weighted sum Γ_ϕ to the concatenated matrix, leading to the final assignment matrix $\mathbf{S}^{(i)}$.

$$\mathbf{S}^{(i)} = \Gamma_\phi \left(\Big\|_{p=0}^{|h|} \mathbf{S}^{(i)}_{p} \right) \tag{12}$$

Differentiable Classifer

What this stage do is to map the graph-level embedding vector $\mathbf{X}^{(i)}_{final}$ to a specific predicted class number $\hat{y}$. We apply a standard multi-layer fully connected neural network to transform the dimension of the graph-level embedding to the number of classes and the predicted class number is compared against the ground-truth label using the following loss function:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j} log \hat{y}_{i,j} \tag{13}$$

where N is the set of training samples, M denotes the number of classes, y is the true label and $\hat{y}$ is the value predicted by the model.

Experiments

In this section, we conduct extensive experiments on 10 benchmark datasets for MTS classification, and compare the results of our model (MTPool) with other 10 baselines.

Experiment Settings

Datasets We use 10 publicly available benchmark datasets from UEA MTS classification archive[1]. The main character-

[1]Datasets are available at http://timeseriesclassification.com. We exclude data with extremely long length, unequal length, high dimension size and in-balance split.

Table 2: Accuracy of the 14 algorithms on the default training/test datasets of 10 selected UEA MTSC archives.

Methods / Datasets	AF	FM	HMD	HB	LIB	MI	NATO	PD	SRS2	SWJ	Wins
ED	0.267	0.519	0.279	0.620	0.833	0.510	0.850	0.973	0.483	0.333	0
DTW_I	0.267	0.513	0.297	0.659	0.894	0.390	0.850	0.939	0.533	0.200	0
DTW_D	0.267	0.529	0.231	0.717	0.872	0.500	0.883	0.977	0.539	0.200	0
ED (norm)	0.200	0.510	0.278	0.619	0.833	0.510	0.850	0.973	0.483	0.333	0
DTW_I (norm)	0.267	0.520	0.297	0.658	0.894	0.390	0.850	0.939	0.533	0.200	0
DTW_D (norm)	0.267	0.530	0.231	0.717	0.870	0.500	0.883	0.977	0.539	0.200	0
WEASEL+MUSE	0.400	0.550	0.365	0.727	0.894	0.500	0.870	0.948	0.460	0.267	0
HIVE-COTE	0.133	0.550	0.446	0.722	**0.900**	0.610	0.889	0.934	0.461	0.333	1
MLSTM-FCN	0.333	0.580	**0.527**	0.663	0.850	0.510	0.900	0.978	0.472	0.400	1
TapNet	0.200	0.470	0.338	**0.751**	0.878	0.590	0.939	0.980	0.550	0.133	1
MTPool-M	**0.533**	0.504	0.486	0.742	0.828	0.560	0.928	0.978	0.550	0.533	1
MTPool-D	0.400	0.530	0.459	0.737	0.811	0.600	**0.944**	0.977	0.550	0.533	1
MTPool-S	0.400	0.590	0.473	0.722	0.811	0.540	0.889	**0.983**	0.539	**0.667**	1
MTPool-One	0.400	0.570	0.405	0.717	0.833	0.540	0.889	0.970	0.539	0.600	0
MTPool-Corr	0.400	0.590	0.419	0.722	0.828	0.560	0.904	0.973	0.550	0.600	0
MTPool	0.467	**0.620**	0.432	0.742	0.861	**0.630**	0.904	**0.983**	**0.600**	**0.667**	**4**

istics of each dataset is summarized in Table 1.

Metrics For each dataset, we compute the classification accuracy as the evaluation metric.

Methods for Comparison We use the following implementations of the MTS classifiers:

- **ED, DTW_I, DTW_D, - with and without normalization (norm)** (Bagnall et al. 2018): 1-Nearest Neighbour with distance functions: Euclidean (ED); dimension-independent dynamic time warping (DTW_I); and dimension-dependent dynamic time warping (DTW_D).
- **WEASEL+MUSE** (Schäfer and Leser 2017): The Word extraction for time series classification (WEASEL) with a Multivariate Unsupervised Symbols and dErivatives (MUSE) is the most effective bag-of-patterns algorithm for MTSC.
- **HIVE-COTE** (Bagnall et al. 2020) is a heterogeneous meta ensemble for time series classification. This approach is a good baseline for assessing bespoke Multivariate Time Series Classification.
- **MLSTM-FCN** (Karim et al. 2019) uses a LSTM layer and a stacked CNN layer along with squeeze-and-excitation blocks to generate latent features.
- **TapNet** (Zhang et al. 2020) draws on the strengths of both traditional and deep learning approaches. It also constructs a LSTM layer and a stacked CNN layer, followed by an attentional prototype network.
- **MTPool-M** is the MTPool framework with MemPool.
- **MTPool-D** is the MTPool framework with DiffPool.
- **MTPool-S** is the MTPool framework with SAGPool.
- **MTPool-One** is the MTPool framework with Variational Pooling and all-one adjacency matrix.
- **MTPool-Corr** is the MTPool framework with Variational Pooling and correlation coefficient adjacency matrix.
- **MTPool** is the MTPool framework with our proposed Variational Pooling and dynamic adjacency matrix.

Training Details All the networks are implemented with Pytorch 1.4.0 in python 3.6.2, and trained with 10000 epochs (computing infrastructure: Ubuntu 18.04 operating system, GPU NVIDIA GeForce RTX 2080 Ti with 8 Gb GRAM and 32 Gb of RAM). The size of the three convolutional kernels are set to be $\{3, 5, 7\}$ respectively and the channels are chosen from $\{1, 5, 10\}$. The threshold to make the adjacency matrix sparse is chosen from $\{0.05, 0.1, 0.2\}$ and the output dimension of the GNNs layer is 128. For the pooling layers, the heads of the centroids are chosen from $\{1,2,4\}$, the reduce factor is chosen from $\{2, 3, 6\}$ and the number of nodes in the final pooling layer is 1. The initial learning rate is 0.00001 and the categorical cross-entropy loss and the Adam optimization are used to optimize the parameters of our models.

Main Results

Tabel 2 shows the accuracy results on the selected 10 UEA datasets of MTPool and other MTS classifiers. For some approach that ran out of memory we refer to (Ruiz, Flynn, and Bagnall 2020). The highest accuracy score in each dataset is bolded.

First of all, we can observe that MTPool wins on four datasets, better than several other advanced MTS classification methods, (such as TapNet or MTPool-D wins on one dataset). More importantly, with the different datasets, the

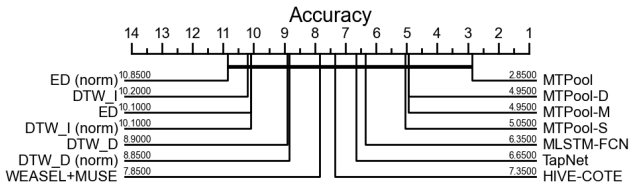

Figure 2: Critical difference plot of the MTS classifiers on the 10 selected UEA datasets with alpha equals to 0.05.

number of variables and lengths span a large range. For example, the number of variables ranges from 2 to 64 and the length ranges from 8 to 300. In the case of variable dataset parameters, our framework maintains considerable competitiveness, which also proves the robustness of our model.

More specifically, when the number of variables in the datasets is relatively large, MTPool can handle this by increasing the number of pooling layers and make the graph pooling process more hierarchical, thus performing well. For instance, MTPool achieved the best and the second best accuracy on the Heartbeat (61 variables) and MotorImagery (64 variables) respectively. This means that when the number of variables is relatively large, the hierarchical pooling framework we provide can better capture the structure of the graph and generate time series embeddings with stronger characterization capabilities. It is worth noting that when the number of variables is relatively small, by reducing the number of pooling layers, MTPool can also achieve good accuracy: MTPool obtains the best accuracy on PenDigits (2 variables), SelfRegulationSCP2 (7 variables) and StandWalkJump (4 variables) datasets.

We also conducted a statistical test to evaluate the performance of MTPool compared to the other MTS classifiers. We present in Figure 2 the critical difference plot with alpha equals to 0.05 from results shown in Table 2. The values correspond to the average rank and the classifiers linked by a bar do not have a statistically significant difference. This figure illustrates the effectiveness of the MTPool framework: even if different pooling methods are used, good results can be achieved on these datasets. MTPool, MTPool-M, MTPool-D and MTPool-S are ranked 2.85, 4.95, 4.95 and 5.05 respectively. Their average rankings surpass other state-of-the-art models, which also inspired us to use graphs to model multivariate time series and better characterize the relationships among different variables. This figure also reflects that for MTS classification problems, some traditional methods, such as ED, DTW$_I$, and DTW$_D$ with or without normalization perform generally, and some deep learning methods (TapNet or MLSTM-FCN) proposed in recent years generally surpass these traditional methods.

Ablation Study

First, we substitute our Variational Pooling layers with some other advanced pooling methods in MTPool framework. Then we use static adjacency matrix for initial graph structure instead of dynamic adjacency matrix. Table 2 shows the comparison results. The important conclusions of these results are as follows:

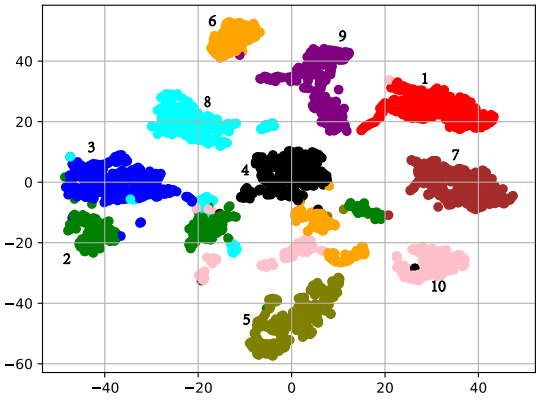

Figure 3: Class Prototype Inspection: visualize the 128-dimension multivariate time series embeddings in a two dimensional image by t-SNE.

- Different hierarchical graph pooling methods can be incorporated in our MTPool framework and can achieve comparable or better performance than state-of-the-art MTSC methods.
- Different adjacency matrices can be used in our MTPool framewor. However, the performance of the all-one matrix is slightly worse than the correlation coefficient matrix, and our dynamic matrix achieves the best.
- Compared with other state-of-the-art graph pooling methods, our proposed Variational Pooling can achieve the best performance in most cases.

Inspection of Class Prototype

In this section, we visualize the class prototype and its corresponding time series embedding to prove the effectiveness of our well-trained time series embedding. We use the t-SNE algorithm (Maaten and Hinton 2008) to visualize the 128-dimensional time series embedded in the form of two-dimensional images. We use different colors to distinguish different categories. Figure 3 shows the embeddings learned for the PenDigits dataset, which contains 3498 test samples in 10 different categories. The following conclusions can be drawn from the results: 1) The distance between data samples from different categories is much greater than the distance between data samples from the same category, which means that we can easily use the learned multivariate time series to embed the time sequence classification; 2) Low-dimensional time series embedding provides us with a more interpretable perspective to understand the problem of classifiers. For example, we can see that category 2, category 6, and category 10 are not divided into a complete piece. In fact, these three categories are all divided into several subparts. It really helps to identify problems with the classifier and take further measures, such as adding more training samples in these three classes.

Conclusion

In this paper, we propose a novel MTS classification framework, MTPool. MTPool is the first graph pooling based framework for MTSC, which can model the pairwise dependencies among variables in MTS explicitly and attain global embedding with strong interpretability and expressiveness. Experimental results demonstrate our model achieves state-of-the-art performance in the existing models.

For future research, it's promising to explore and design more powerful hierarchical graph pooling methods which can be incorporated into our MTPool framework to attain more expressive and interpretable global representation for MTS.

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. J.; Flynn, M.; Large, J.; Lines, J.; and Middlehurst, M. 2020. A tale of two toolkits, report the third: on the usage and performance of HIVE-COTE v1. 0. *CoRR* .

Chen, J.; Ma, T.; and Xiao, C. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247* .

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In Lee, D. D.; Sugiyama, M.; Luxburg, U. V.; Guyon, I.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 29*, 3844–3852. Curran Associates, Inc.

Gao, H.; and Ji, S. 2019. Graph u-nets. *arXiv preprint arXiv:1905.05178* .

Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017. Inductive Representation Learning on Large Graphs. *CoRR* abs/1706.02216.

Kang, H.; and Choi, S. 2014. Bayesian common spatial patterns for multi-subject EEG classification. *Neural Networks* 57: 39–50.

Karim, F.; Majumdar, S.; Darabi, H.; and Harford, S. 2019. Multivariate LSTM-FCNs for time series classification. *Neural Networks* 116: 237–245.

Khasahmadi, A. H.; Hassani, K.; Moradi, P.; Lee, L.; and Morris, Q. 2020. Memory-Based Graph Networks. In *International Conference on Learning Representations*.

Kipf, T. N.; and Welling, M. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *CoRR* abs/1609.02907.

Lee, J.; Lee, I.; and Kang, J. 2019. Self-attention graph pooling. *arXiv preprint arXiv:1904.08082* .

Maaten, L. v. d.; and Hinton, G. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9(Nov): 2579–2605.

Morris, C.; Ritzert, M.; Fey, M.; Hamilton, W. L.; Lenssen, J. E.; Rattan, G.; and Grohe, M. 2019. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 4602–4609.

Pei, W.; Dibeklioğlu, H.; Tax, D. M.; and van der Maaten, L. 2017. Multivariate time-series classification using the hidden-unit logistic model. *IEEE transactions on neural networks and learning systems* 29(4): 920–931.

Ranjan, E.; Sanyal, S.; and Talukdar, P. P. 2020. ASAP: Adaptive Structure Aware Pooling for Learning Hierarchical Graph Representations. In *AAAI*, 5470–5477.

Ruiz, A. P.; Flynn, M.; and Bagnall, A. 2020. Benchmarking Multivariate Time Series Classification Algorithms. *arXiv preprint arXiv:2007.13156* .

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2008. The graph neural network model. *IEEE Transactions on Neural Networks* 20(1): 61–80.

Schäfer, P.; and Leser, U. 2017. Multivariate time series classification with WEASEL+ MUSE. *arXiv preprint arXiv:1711.11343* .

Seto, S.; Zhang, W.; and Zhou, Y. 2015. Multivariate time series classification using dynamic time warping template selection for human activity recognition. In *2015 IEEE Symposium Series on Computational Intelligence*, 1399–1406. IEEE.

Sun, C.; Chen, C.; Li, W.; Fan, J.; and Chen, W. 2019. A Hierarchical Neural Network for Sleep Stage Classification Based on Comprehensive Feature Learning and Multi-Flow Sequence Learning. *IEEE Journal of Biomedical and Health Informatics* 24(5): 1351–1366.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* .

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 (IJCNN)*, 1578–1585. IEEE.

Wu, Z.; Pan, S.; Long, G.; Jiang, J.; Chang, X.; and Zhang, C. 2020. Connecting the Dots: Multivariate Time Series Forecasting with Graph Neural Networks. *arXiv preprint arXiv:2005.11650* .

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018. How Powerful are Graph Neural Networks? *ArXiv* abs/1810.00826.

Ying, Z.; You, J.; Morris, C.; Ren, X.; Hamilton, W.; and Leskovec, J. 2018. Hierarchical graph representation learning with differentiable pooling. In *Advances in neural information processing systems*, 4800–4810.

Yu, Z.; and Lee, M. 2015. Real-time human action classification using a dynamic neural model. *Neural Networks* 69: 29–43.

Zhang, X.; Gao, Y.; Lin, J.; and Lu, C.-T. 2020. TapNet: Multivariate Time Series Classification with Attentional Prototypical Network. In *AAAI*, 6845–6852.