Contrastive Learning for Time Series on Dynamic Graphs

Yitian Zhang
Electrical and Computer Engineering
McGill University
Montreal, Canada
yitian.zhang@mail.mcgill.ca

Florence Regol
Electrical and Computer Engineering
McGill University
Montreal, Canada
florence.robert-regol@mail.mcgill.ca

Antonios Valkanas
Electrical and Computer Engineering
McGill University
Montreal, Canada
antonios.valkanas@mail.mcgill.ca

Mark Coates
Electrical and Computer Engineering
McGill University
Montreal, Canada
mark.coates@mcgill.ca

***Abstract*—There have been several recent efforts towards developing representations for multivariate time-series in an unsupervised learning framework. Such representations can prove beneficial in tasks such as activity recognition, health monitoring, and anomaly detection. In this paper, we consider a setting where we observe time-series at each node in a dynamic graph. We propose a framework called *GraphTNC* for unsupervised learning of joint representations of the graph and the time-series. Our approach employs a contrastive learning strategy. Based on an assumption that the time-series and graph evolution dynamics are piecewise smooth, we identify local windows of time where the signals exhibit approximate stationarity. We then train an encoding that allows the distribution of signals within a neighborhood to be distinguished from the distribution of non-neighboring signals. We first demonstrate the performance of our proposed framework using synthetic data, and subsequently we show that it can prove beneficial for the classification task with real-world datasets.**

***Index Terms*—Contrastive learning, representation learning, time series, dynamic graphs**

I. Introduction

Time series constitute a challenging data type for modeling, especially for supervised learning, due to their sparse labeling and complexity. To address this challenge, we can employ unsupervised methods to learn embeddings of the time series, and thereby extract informative low-dimensional representations. These general representations of the input data, derived without any need for labels, can be used for any downstream task.

In the last several years, self-supervised learning (SSL) has emerged as an effective strategy for learning representations. One form of SSL is contrastive learning, popularized by the SimCLR approach in [1]. One danger with self-supervised learning is *collapse*, when the model learns to output similar or even identical embeddings for all samples. Contrastive learning avoids collapse by identifying positive and negative training pairs. The embeddings of samples in a positive pair are encouraged to be similar, while those of samples in a negative pair are pushed apart.

Several approaches have emerged for contrastive learning for time series. Contrastive Predictive Coding (CPC) [2] is an effective strategy that first compresses high-dimensional data into a compact latent embedding space and then uses autoregressive models to predict the subsequent values of the signals. It uses predictive coding principles to train the encoder on a probabilistic contrastive loss. Franceschi et al. employ a triplet loss in [3], which strives to ensure that a reference time series has a representation that is close to any one of its subseries (a positive sample) but far from negative series (chosen at random). Temporal Neighborhood Coding (TNC) takes advantage of the local smoothness of the signals to learn generalizable representations for windows of a time series [4]. This is achieved by ensuring that in the representation space, the distribution of signals that are close together in time is distinguishable from the distribution of signals that are far apart. TNC also takes into account the possibility that a pair of negative samples may also be similar.

Compared to contrastive learning, non-contrastive approaches are conceptually simple, and do not need a large batch size or a large memory bank to store negative samples. Notable approaches include Bootstrap Your Own Latent (BYOL) [5] and Simple Siamese (SimSiam) [6]. These methods train a student network to predict the representations of a teacher network. The weights of the latter are a moving average of the student's weights, or are shared with the student, but no gradient is backpropagated through the teacher. Recent efforts have explored the development of more effective loss terms. For example, Variance-Invariance-Covariance Regularization (VICReg) in [7] improves and builds upon the Barlow Twins loss of [8].

Some supervised learning frameworks have considered learning a graph structure to capture correlations in multivariate time series. For example, in [9], Hu et al. propose

EvoNet, which constructs a dynamic graph from time series data and can be used for event prediction. However, unsupervised representation learning of time-series on graphs remains underexplored in the literature.

In this paper, we propose a framework called *GraphTNC* for learning joint representations of the graph and the time-series. This procedure is designed for the setting where the underlying states of the signals and graph change over time. This model is also scalable to time-series with a static graph, where the graph input at each time-step are the same. We assess the quality of the learned representations on two datasets and show that the representations are general and transferable to downstream tasks such as classification.

Our contributions can be summarized as follows:

- We propose a novel encoder for learning representations of multivariate time series data on dynamic or static graphs, through a contrastive learning framework.
- We generalize non-contrastive learning methods from the computer vision domain to address non-stationary multivariate time series data.

II. Problem Setting

We consider the task of unsupervised learning of representations for time series on graphs. We denote a multivariate time series as $\mathbf{X} \in \mathbb{R}^{N \times T}$ where N is the number of univariate time series, and T is the total length of the time series. A window of fixed length w starting at time index t is contained by the $[t, t+w]-$th columns of $\mathbf{X}$: $\mathbf{X}_{[t,t+w]} \in \mathbb{R}^{N \times w}$, and is denoted by $\mathbf{X}^t$. w is assumed to be constant so we do not include it in the notation and only specify it in the text when essential for clarity. Associated with the multivariate time series, we also have dynamic graphs of N nodes whose edges evolve alongside the time series. Each of the N univariate time series is associated with one node in the graph. The edges between nodes are assumed to be indicative of the evolving correlation structure. Analogously to the windows for the time series, we denote a window of dynamic graphs by $\boldsymbol{\mathcal{G}}^t = [\mathcal{G}_t, \dots, \mathcal{G}_{t+w}]; \mathcal{G}_i = (\mathcal{V}, \mathcal{E}_i), |\mathcal{V}| = N$ where each graph $\mathcal{G}_i$ is associated with the state of the graph at time i. Only the edge set $\mathcal{E}$ is indexed as the node set $\mathcal{V}$ stays constant. The goal is to learn a representation $\mathbf{z}^t \in \mathbb{R}^h$ of a time series window and its associated graphs $(\mathbf{X}^t, \boldsymbol{\mathcal{G}}^t)$: $f^{enc}(\mathbf{X}^t, \boldsymbol{\mathcal{G}}^t) = \mathbf{z}^t$, where h is the dimension of the joint representations. In this work, we subsequently use this representation to perform classification.

III. Methodology

We design an architecture that constructs a representation of a window of a multivariate time series and an associated sequence of graphs. The architecture is an encoder consisting of two modules. In the ensuing subsections we describe these modules and the loss function used to train the encoder.

A. Encoder $f^{enc}(\mathbf{X}^t, \boldsymbol{\mathcal{G}}^t)$

Our encoding approach can be decomposed into two main building blocks : a) a *static graph encoding module* that learns the state of the graph and its relationship with the multivariate time series; and b) a *temporal module* which captures the dynamics of the data.

a) Static graph encoding module: The purpose of this module is to learn the relationships between the node embeddings and the multivariate signal at a timestep i. To do so, we first need a representation of each individual node based on the state of the graph $\mathcal{G}_i$. This can be provided by any node embedding function $f^{\mathcal{G}}$ that takes a graph as input:

$$\mathbf{H}_i = f^{\mathcal{G}}(\mathcal{G}_i), \quad \mathbf{H}_i \in \mathbb{R}^{N \times k}, \tag{1}$$

where k is the dimension of the output node embeddings. Next, we concatenate $\mathbf{H}_i$ with the time series of this timestep denoted by $\mathbf{x}_i \in \mathbb{R}^N$ and pass it through a neural network to obtain $\mathbf{e}_i$, the final representation of the graph-signal interaction at timestep i:

$$\mathbf{e}_i = NN^1([\text{vec}(\mathbf{H}_i)||\mathbf{x}_i]), \quad \mathbf{e}_i \in \mathbb{R}^d, \tag{2}$$

where d is the dimension of the graph-signal interaction representation, $\text{vec}(\cdot)$ is an operator that stacks the columns of a matrix, and $[\cdot||\cdot]$ denotes the concatenation of two vectors ($\text{vec} : \mathbb{R}^{a\times b} \to \mathbb{R}^{ab}$, $[\cdot||\cdot] : \mathbb{R}^a||\mathbb{R}^b \to \mathbb{R}^{a+b}$).

b) Temporal Module: We use a temporal-based neural network f^{temp} to capture the dynamic nature of the data $(\mathbf{X}^t, \boldsymbol{\mathcal{G}}^t)$ and to obtain the final representation $\mathbf{z}^t$. The network f^{temp} outputs the hidden state of the next timestep $\mathbf{s}_{i+1} \in \mathbb{R}^s$ based on the current hidden state $\mathbf{s}_i \in \mathbb{R}^s$ and on the input at time i. In our framework, the input is constructed of the signal $\mathbf{x}_i$ concatenated with the processed graph-signal interaction $\mathbf{e}_i$. The final representation $\mathbf{z}_t$ is obtained by passing the last hidden state of the window $\mathbf{s}_w$ through a neural network:

$$\mathbf{s}^t_{i+1} = f^{temp}([\mathbf{x}_{t+i}||\mathbf{s}^t_i||\mathbf{e}_{t+i}]) \tag{3}$$

$$\mathbf{z}^t = NN^2(\mathbf{s}^t_w) \tag{4}$$

We use a 1-layer graph convolution as $f^{\mathcal{G}}$ and a 1-layer bi-directional Gated Recurrent Unit (GRU) as f^{temp}. Both NN^1 and NN^2 are 1-layer feed-forward neural networks (FNN).

B. Loss function

We define a discriminator $D(\mathbf{z}^t, \mathbf{z})$. The objective function is to make the probability likelihood estimation of the discriminator to be close to 1 if $\mathbf{z}$ and $\mathbf{z}^t$ are representations of neighboring windows, and close to 0 otherwise. Following [4], we view windows that are close in time as neighboring windows, and use the Augmented Dickey-Fuller (ADF) statistical test to find the neighborhood range. The neighbourhood $\mathcal{N}^t$ is selected based only on time series, since the underlying states of graphs and signals are assumed to evolve together.

The loss function is defined as:

$$\mathcal{L}(\mathbf{X}^t, \boldsymbol{\mathcal{G}}^t) = -\mathbb{E}_{(\mathbf{X}^l, \boldsymbol{\mathcal{G}}^l) \sim \mathcal{N}^t}\Big[\log \mathcal{D}(\mathbf{z}^t, \mathbf{z}^l)\Big] + \mathbb{E}_{(\mathbf{X}^k, \boldsymbol{\mathcal{G}}^k) \sim \bar{\mathcal{N}}^t}\Big[(1-m)\log(1 - \mathcal{D}(\mathbf{z}^t, \mathbf{z}^k)) + m \log \mathcal{D}(\mathbf{z}^t, \mathbf{z}^k)\Big] \tag{5}$$

where m is the probability of sampling a positive window from the non-neighboring region $\bar{\mathcal{N}}^t$. By optimizing this function, representations $\mathbf{z}^l = f^{enc}(\mathbf{X}^l, \boldsymbol{\mathcal{G}}^l)$ of samples from a neighborhood $(\mathbf{X}^l,\ \boldsymbol{\mathcal{G}}^l) \in \mathcal{N}^t$, can be distinguished from representations $\mathbf{z}^k = f^{enc}(\mathbf{X}^k, \boldsymbol{\mathcal{G}}^k)$ of samples from outside the neighborhood.

IV. Experiments

In the experiments, we evaluate the performance of our proposed model on a synthetic dataset in a controlled setting, and on a real-world dataset. Both datasets have underlying states that change over time. Therefore, a state is associated with each time window $(\mathbf{X}^t,\ \boldsymbol{\mathcal{G}}^t)$. The performance of the learned representations $\mathbf{z}$ is evaluated by the downstream classification task, where the states are the classification targets. Below we describe the datasets, experiment setup and results in details.

A. Datasets

1) Synthetic data: The synthetic dataset contains a multivariate time series influenced by a dynamic graph, that is also synthetically generated. The generation of both the time series and the graphs is driven by an underlying state of the time series that is modeled by a Hidden Markov Model (HMM). In each state, the time series are generated from a different generative process including Nonlinear Auto-regressive Moving Average models with different sets of parameters and Gaussian Processes with different kernel functions, in a similar fashion to that employed in the synthetic experiment in [4]. The features at each time step are concatenated in a vector $\mathbf{f}_t \in \mathbb{R}^N$.
For the graph structure, each state has a different initial random graph $\mathcal{G}_0^s$ generated from an Erdős–Rényi model with probability p^s of an edge between nodes, where s is the state number. With a probability $q^s = p^s/10(1-p^s)$, $\mathcal{G}_{t-1}^s$ adds new edges or removes existing ones to generate $\mathcal{G}_t^s$.
The time series data is generated by:

$$\mathbf{x}_{t+1} = rA_t\mathbf{f}_t + (1-r)\mathbf{f}_t, \quad 0 \le r \le 1, \tag{6}$$

where A_t is the adjacency matrix of $\mathcal{G}_t$, and r weights how much influence the graph has on the time series.

2) EEG: EEG signals are recorded from probes connected to brains of human subjects. The dataset, which originates from an online data science competition[1], contains 32 channels of EEG recordings of subjects performing hand grasping and lifting actions. Each hand action is divided into 7 states that include: the initial movement, first touch of the object to be lifted, start of loading phase, hand lift off, change of hands, release and no action. The graph structure for this dataset encodes the spatial relationship between the 32 electrode locations. The dataset provides a map of the physical locations of the electrode probes with respect to the brain. These probes are arranged in a grid. We define a graph where each node represents an electrode probe and an edge connects two probes if they are direct neighbors on the grid. Since the graph is static, it is repeated at each time step to fit our model. We extract 100 signals of length 60 timesteps to train our model.

[1]https://www.kaggle.com/c/grasp-and-lift-eeg-detection/data

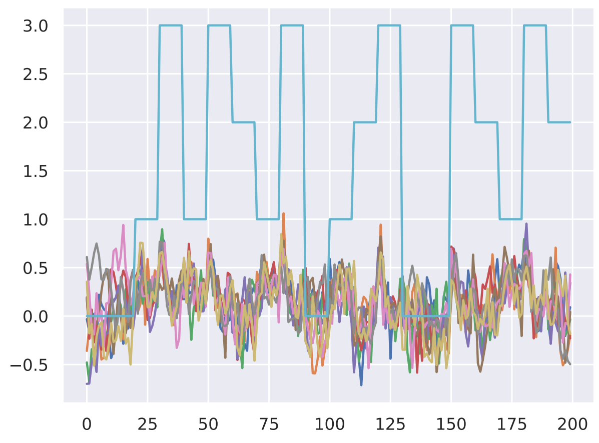


Fig. 1: Synthetic dataset: Light blue discrete signal represents underlying state. Continuous signals represent the N features.

TABLE I: Hyper-parameters of GraphTNC

	Synthetic	**EEG**
Graph node (feature) number N	10	32
Graph encoding size k	4	4
Graph-signal interaction size d	8	32
GRU input size $(N+d)$	18	64
GRU hidden size	64	64
Joint representations size h	8	32

B. Experiments Setup

1) GraphTNC vs baseline TNC: In this experiment, we compare the classification performance of the representations learned from time series with and without considering the graph. For fair comparison, we use the same encoder proposed for the baseline TNC [4] as f^{temp}, which is a 1-layer bi-directional GRU. In our architecture, we add a graph encoding module $f^{\mathcal{G}}$ (a 1-layer graph convolution) before the temporal module, such that the input of f^{temp} is the combined information of the graph and signal. We also followed the setting $m = 0.05$ in (5) from [4] which is the probability of positive window in non-neighboring region. Detailed hyperparameters can be seen in Table I.

For training, we use the Adam optimizer with a learning rate of $1e{-}3$ and a weight decay of $1e{-}5$ and train for 100 epochs with early stop for both datasets. As stated in Section III, although the discriminator and encoder are learned together during the training phase, only the encoder is required during inference. The window size w is selected through experiments such that it is long enough to contain information of underlying state but not too long to span over multiple states, following the same rationale in TNC [4].

To evaluate the quality of the representations, we use classification as a downstream task. The classifier is a 1-layer FNN on top of the frozen representation with h as input dimension and S as output dimension, trained on cross-entropy loss. We use the simple structure to reduce the impact of

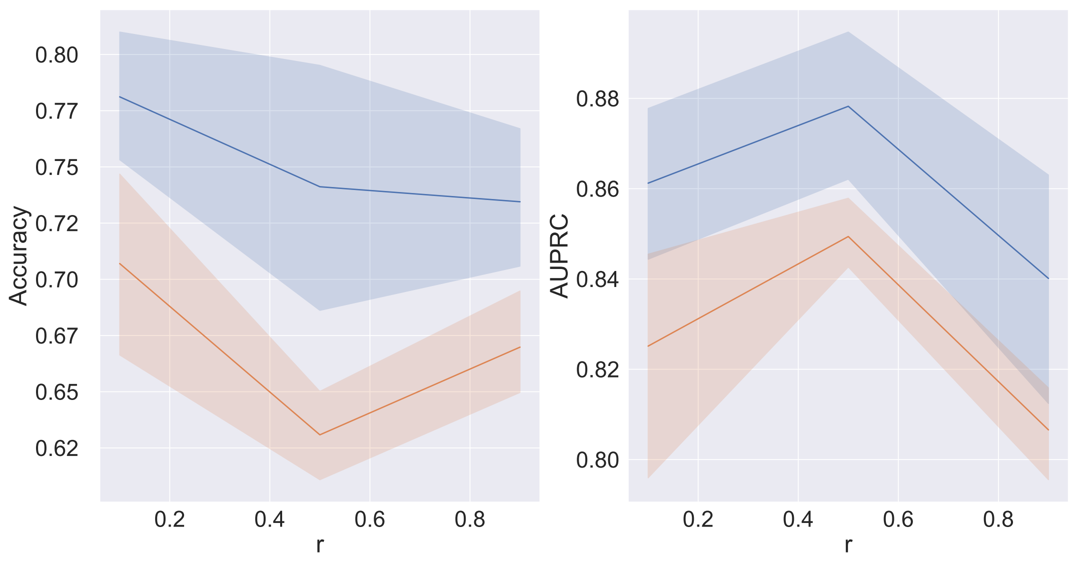


Fig. 2: Results (with 95% confidence intervals shaded) of synthetic dataset experiment with various r values from eq.(6). Left: Accuracy results. Right: Area under precision recall curve (AUPRC). Our GraphTNC is shown in blue, baseline is in orange.

the classifier on the final results. The performance is reported as the prediction accuracy and the area under the precision-recall curve (AUPRC) score since AUPRC is a more accurate reflection of model performance for imbalanced data.

2) GraphTNC vs non-contrastive learning: Throughout this experiment, we retain our proposed encoder $f^{enc}(\mathbf{X}^t, \boldsymbol{\mathcal{G}}^t)$, and compare the performance of the GraphTNC contrastive learning approach with two non-contrastive learning methods, BYOL [5] and SimSiam [6]. BYOL has an asymmetric architecture where the weights θ_m of one encoder are an exponential moving average of the other encoder's weights θ. A predictor g with weights ϕ is used in the branch with learnable weights. SimSiam uses a predictor on one branch and a stop-gradient operation on the other. In the original papers [5], [6], the inputs to the two encoders are the original image and an augmentation.

To generalize these methods to our setting, we feed $(\mathbf{X}^t,\ \boldsymbol{\mathcal{G}}^t)$ and $(\mathbf{X}^l,\ \boldsymbol{\mathcal{G}}^l) \in \mathcal{N}^t$ as positive pairs to the student and teacher networks. Non-neighborhood samples are not required. For the projector and predictor for both BYOL and SimSiam, we use a 2-layer FNN with size 128-128. We then compare the performance of all unsupervised approaches to a supervised model where the classifier and the encoder are trained end-to-end. For the supervised setting, the architectures of the encoder and the classifier are the same as for the unsupervised models. We evaluate the performance of the representations learned from different methods via the state classification accuracy and AUPRC.

C. Experiments results and discussion

1) GraphTNC vs baseline TNC: Table II presents the state classification results. An asterisk indicates a statistically significant difference at the 5% level between the GraphTNC and the baseline for a Wilcoxon signed-rank test. First, we can observe that our encoder, which learns joint representations of the time series and of the graphs, consistently outperforms the baseline with significance most of the time for both the simulated data and EEG dataset, with the same order of parameterization. Therefore, we conclude that modeling the relation between the features of a time series can lead to improvement in performance.

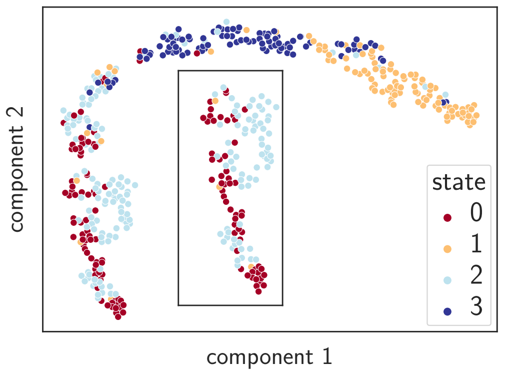
TABLE II: Classification results. The asterisk represents statistically significant result using paired Wilcoxon test ($p < 0.05$).

	GraphTNC (ours)		**TNC**	
Dataset	***AUPRC***	***Accuracy***	***AUPRC***	***Accuracy***
$r = 0.1$	**0.86±0.03***	**0.78±0.06***	0.83±0.04	0.71±0.06
$r = 0.5$	**0.88±0.03***	**0.74±0.09***	0.85±0.01	0.63±0.03
$r = 0.9$	**0.84±0.04***	**0.73±0.05***	0.81±0.04	0.61±0.02
EEG	0.54±0.02	**0.92±0.02**	0.54±0.02	0.90±0.03

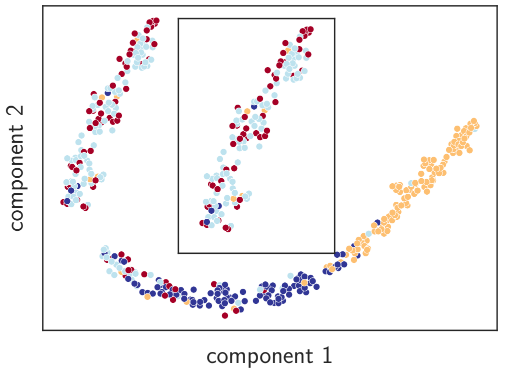
Second, to further understand how the role of an underlying graph influences the performance, we generate multiple synthetic datasets based on different r parameter values in Equation (6). A larger r represents that the time series data is more dependent on the graph-defined spacial filtering operation of Eq. (6). We conduct the experiment for $r \in \{0.1, 0.5, 0.9\}$. We evaluate the model performance by training different models on 10 splits for each r value and report the AUPRC. Our proposed method GraphTNC (in blue) consistently outperforms the time series baseline TNC (in orange). This is true for both the accuracy metric and the AUPRC. The use of both metrics is important since it accounts for the fact that the label distribution of the synthetic data is not always uniform. AUPRC combines the precision and recall metrics and is known to be a reliable metric for imbalanced datasets. Besides

TABLE III: Classification performance of joint representations of TS and Graph learned from different frameworks.

	Synthetic $r = 0.1$			**EEG**		
	AUPRC	***Accuracy***	*Params*	***AUPRC***	***Accuracy***	*Params*
GraphTNC (Ours)	**0.86±0.03**	**0.78±0.06**	34k	0.54±0.02	**0.92±0.02**	59k
BYOL	0.73±0.10	0.56±0.10	67k	**0.55±0.02**	0.91±0.03	92k
SimSiam	0.74±0.10	0.57±0.10	67k	**0.55±0.03**	0.90±0.03	92k
Supervised	**0.95±0.04**	**0.83±0.09**	34k	**0.57±0.03**	**0.92±0.01**	60k



(a) GraphTNC (ours)



(b) TNC

Fig. 3: T-SNE visualization of the 8-dimensional learned encoding $\mathbf{z}^t$ of windows of simulated dataset with $r = 0.1$. The colors represent the underlying state of the windows. The top figure shows the clustering result of GraphTNC (ours), and the bottom figure shows the results for TNC.

reporting the mean and standard deviation of the results in Table II, we also report 95% confidence intervals in Fig. 2. The 95% confidence intervals are obtained via a non parametric method (bootstrap). We also present a visualisation of the encoding in Fig. 3 for $r = 0.1$. We can see that the GraphTNC representations of are more clearly separated than the TNC representations, especially for states 0 and 2.

2) GraphTNC vs non-contrastive learning: Table III displays the classification performance of the representations obtained from the various approaches on the two datasets. The classifier performance of GraphTNC is closer to the supervised model compared to the two other non-contrastive learning methods. They also have similar parameters since the end-to-end learning framework has the same encoder as GraphTNC, followed by a 1-layer FNN. In the EEG dataset, where the state changes infrequently and the graph is static, BYOL and SimSiam can achieve reasonable results. However, when the non-stationarity increases, such as the synthetic data shown in Fig. 1, the performance drops. Therefore, BYOL and SimSiam, which take neighborhood samples as an augmentation are suitable for more stable time-series scenarios. On the other hand, these two non-contrastive learning approaches need more parameters for training. To conclude, our proposed GraphTNC is an effective approach for learning representations for non-stationary time series on dynamic graphs.

V. Conclusion

We have introduced an unsupervised learning approach called GraphTNC for data consisting of multivariate time-series on dynamic graphs. Our experimental results for the synthetic and real-world datasets show that the methodology is beneficial when the graphs inform or capture the dynamic relations between features in the signals.

References

[1] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proc. Int. Conf. Machine Learning*, 2020, pp. 1597–1607.

[2] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," 2019, arXiv preprint: arXiv 1807.03748.

[3] J.-Y. Franceschi, A. Dieuleveut, and M. Jaggi, "Unsupervised scalable representation learning for multivariate time series," in *Proc. Adv. Neural Info. Process. Syst.*, 2019, pp. 4650–4661.

[4] S. Tonekaboni, D. Eytan, and A. Goldenberg, "Unsupervised representation learning for time series with temporal neighborhood coding," in *Int. Conf. Learning Representations*, 2020.

[5] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar *et al.*, "Bootstrap your own latent-a new approach to self-supervised learning," in *Proc. Adv. Neural Info. Process. Syst.*, 2020, pp. 21 271–21 284.

[6] X. Chen and K. He, "Exploring simple siamese representation learning," in *Proc. IEEE Conf. Comp. Vision and Pattern Recognition*, 2021, pp. 15 750–15 758.

[7] A. Bardes, J. Ponce, and Y. Lecun, "Vicreg: Variance-invariance-covariance regularization for self-supervised learning," in *Proc. Int. Conf. Learning Representations*, 2022.

[8] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny, "Barlow twins: Self-supervised learning via redundancy reduction," in *Proc. Int. Conf. Machine Learning*, 2021, pp. 12 310–12 320.

[9] W. Hu, Y. Yang, Z. Cheng, C. Yang, and X. Ren, "Time-series event prediction with evolutionary state graph," in *Proc. Int. Conf. Web Search and Data Mining*, 2021, pp. 580–588.