GNN4TS

GNNs for Time Series with focus on Dynamic GNNs with application in RS

Klaudia Balcer

Computational Intelligence Research Group, Institute of Computer Science

28 lutego 2024

Uniwersytet Wrocławski

Contents

1. Preliminaries
   - Time Series
   - Graph Neural Networks
   - Sequential Recommendations
   - Encoder-Decoder
2. DGRS [1]
3. DyG2Vec [2]
4. GNN4TS [3]
   - TS Tasks for GNNs
   - Applications
5. Final remarks

Preliminaries

(Multivariate) Time Series

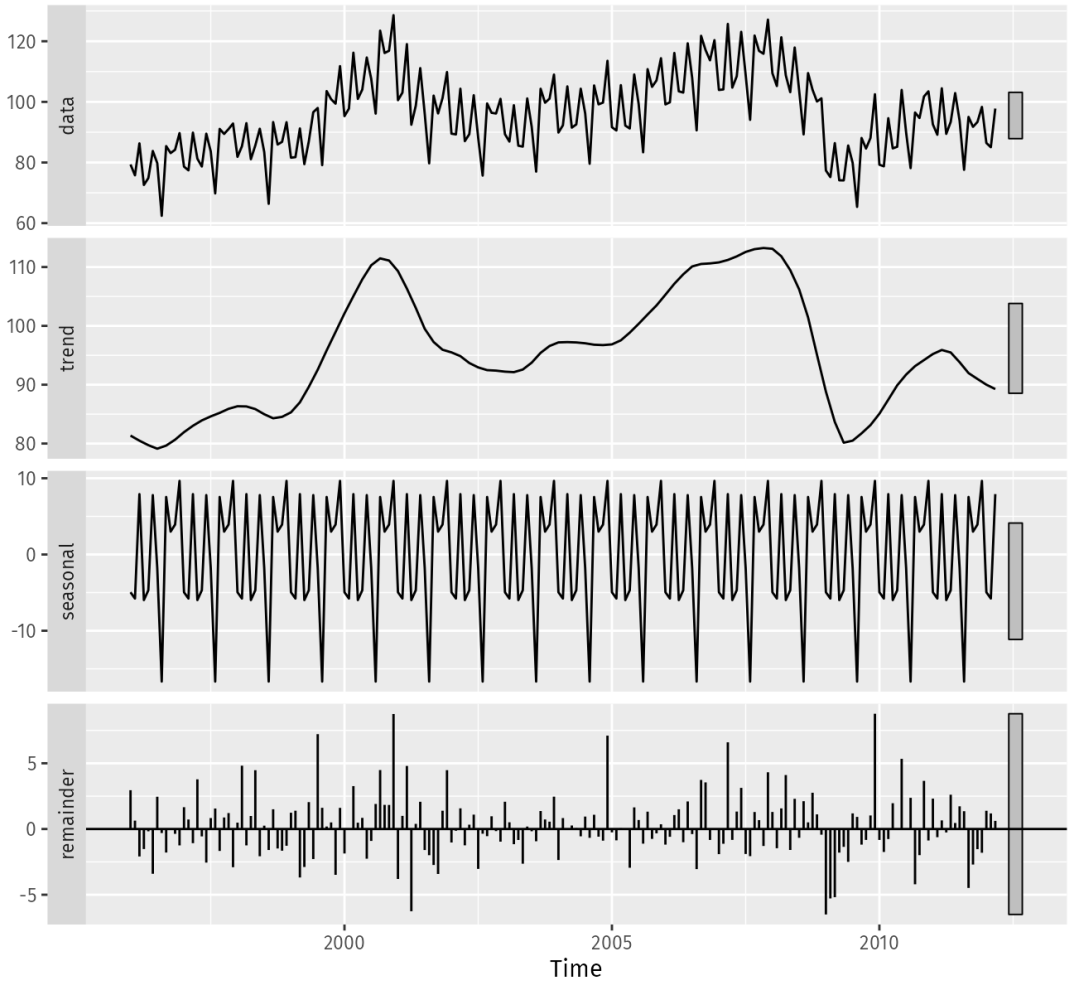
Definition

[3] A **(Multivariate) Time Series** is a sequence of N-dimensional vector observations collected over time, i.e., $X \in \mathbb{R}^{N \times T}$. A regularly sampled multivariate time series has vector observations collected at uniform time intervals, i.e., $x_t \in \mathbb{R}^N$. In an irregularly sampled multivariate time series, there are possibly N unaligned time series with respect to time steps, which implies only $0 \leq n \leq N$ observations available at each time step.

Example models:

- Seasonal Decomposition of Time Series (STL)
- RNN
- Gated Recurrent Unit (GRU)
- Long-Short Term Memory (LSTM)

STL



Rysunek: Original time series, trend, seasonality, noise.

Source: https://otexts.com/fpp2/stl.html

GRU

Initially, for $t = 0$, the output vector is $h_0 = 0$.

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{1}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{2}$$

$$\hat{h}_t = \phi(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \tag{3}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t \tag{4}$$

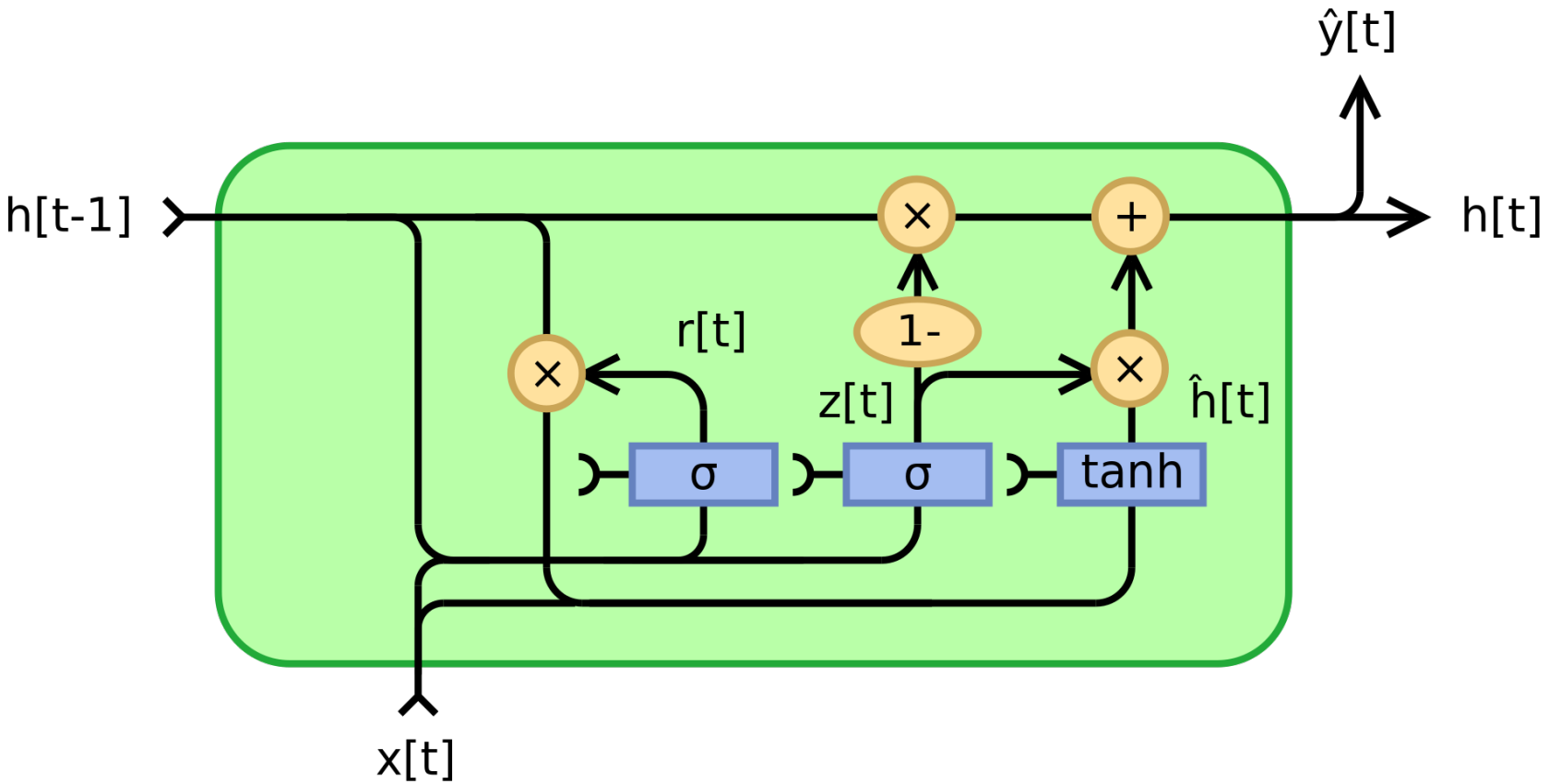
Variables (d denotes the number of input features and e the number of output features):

- $x_t \in \mathbb{R}^d$: input vector
- $h_t \in \mathbb{R}^e$: output vector
- $\hat{h}_t \in \mathbb{R}^e$: candidate activation vector
- $z_t \in (0, 1)^e$: update gate vector
- $r_t \in (0, 1)^e$: reset gate vector
- $W \in \mathbb{R}^{d \times e}$, $U \in \mathbb{R}^{e \times e}$ and $b \in \mathbb{R}^e$: learnable parameters.

Activation functions: σ: The original is a logistic function, ϕ: The original is a hyperbolic tangent. Alternative activation functions are possible, provided that $\sigma(x) \in [0, 1]$.

Source: https://en.wikipedia.org/wiki/Gated_recurrent_unit

GRU



Rysunek: GRU schema

Source: https://en.wikipedia.org/wiki/Gated_recurrent_unit

LSTM

Letting the superscripts d and h refer to the number of input features and number of hidden units, respectively:

- $x_t \in \mathbb{R}^d$: input vector to the LSTM unit
- $f_t \in (0,1)^h$: forget gate's activation vector
- $i_t \in (0,1)^h$: input/update gate's activation vector
- $o_t \in (0,1)^h$: output gate's activation vector
- $h_t \in (-1,1)^h$: hidden state vector also known as output vector of the LSTM unit
- $\tilde{c}_t \in (-1,1)^h$: cell input activation vector
- $c_t \in \mathbb{R}^h$: cell state vector
- $W \in \mathbb{R}^{h \times d}$, $U \in \mathbb{R}^{h \times h}$ and $b \in \mathbb{R}^h$: weight matrices and bias vector parameters which need to be learned during training

Activation functions

- σ_g: sigmoid function.
- σ_c: hyperbolic tangent function.
- σ_h: hyperbolic tangent function or, as the peephole LSTM paper suggests, $\sigma_h(x) = x$.

Source: https://en.wikipedia.org/wiki/Long_short-term_memory

LSTM

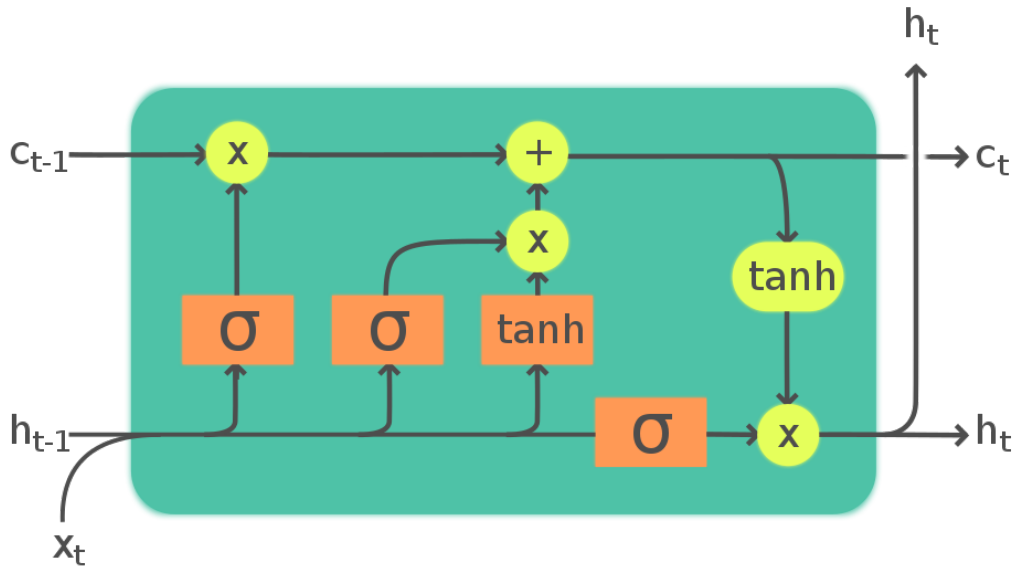
The compact forms of the equations for the forward pass of an LSTM cell with a forget gate are:

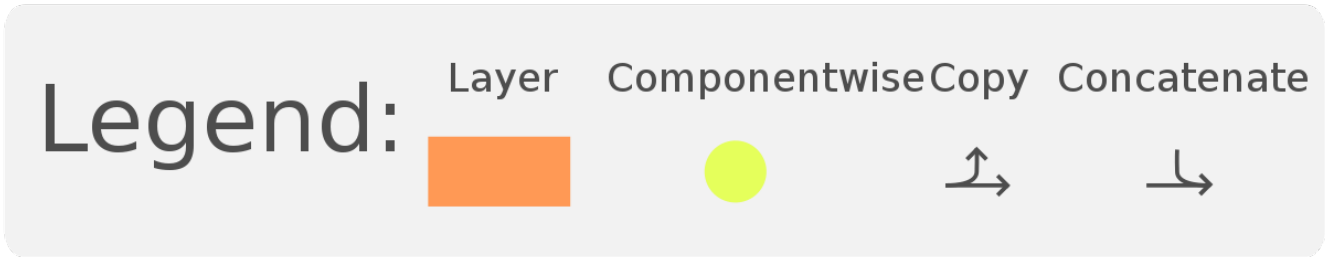
$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \tag{5}$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \tag{6}$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \tag{7}$$
$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \tag{8}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{9}$$
$$h_t = o_t \odot \sigma_h(c_t) \tag{10}$$

where $c_0 = 0$, $h_0 = 0$, $\odot$ denotes the Hadamard product (element-wise product), t indexes the time step.

Source: https://en.wikipedia.org/wiki/Long_short-term_memory

LSTM





Rysunek: LSTM schema.

Source: https://en.wikipedia.org/wiki/Long_short-term_memory

Attributed Graph

Definition

[3] An **Attributed Graph** is a static graph that associates each node with a set of attributes, representing node features. Formally, an attributed graph is defined as $G = (A, X)$, which consists of a weighted) adjacency matrix $A \in \mathbb{R}^{N \times N}$ and a node-feature matrix $X \in \mathbb{R}^{N \times D}$. The adjacency matrix represents the graph topology, which can be characterized by $V = \{v_1, v_2, ..., v_N\}$, the set of N nodes, and $E = \{e_{ij} := (v_i, v_j) \in V \times V | A_{ij} \neq 0\}$, the set of edges; A_{ij} is the (i, j)-th entry in the adjacency matrix A. The feature matrix X contains the node attributes, where the i-th row $x_i \in \mathbb{R}^D$ represents the D-dimensional feature vector of node v_i.

Example models:

- Graph Neural Network (GNN)
- Graph Convolutional Neural Network (GCN)

GNN

For kth layer:

$$a_i^{(k)} = AGGREGATE^{(k)}\left(\left\{h_j^{(k-1)} : v_j \in \mathcal{N}(v_i)\right\}\right),$$

$$h_i^{(k)} = COMBINE^{(k)}\left(h_i^{(k-1)}, a_i^{(k)}\right),$$

where $h_i^{(0)} = x_i$.

Cool intro to GNNs: https://distill.pub/2021/gnn-intro/

Continuous Time-Dynamic Graph

$$G = (V, E, X, T), \text{ where:}$$

- V - nodes (in RS: $V = \mathcal{U} \cup \mathcal{I}$ we have user nodes and item nodes)
- E - edges (in RS: interactions like clicks, bought, opinions)
  $e_i = (u_i, v_i, t_i, m_i)$, where m_i is the vector of edge features.
- $X = (X^V, X^E)$ (node and edge features).
- T - timestamps (in some cases they are considered separatly, in others as a edge property)

In some cases, t can also indicate the order of interactions between two nodes. By recording the time or order of each edge, a dynamic graph can capture the evolution of the relationship between nodes. Dynamic graph embedding aims to learn mapping function $f : V \longrightarrow \mathbb{R}^d$, where d is the number of embedding.

Sequential Recommendations

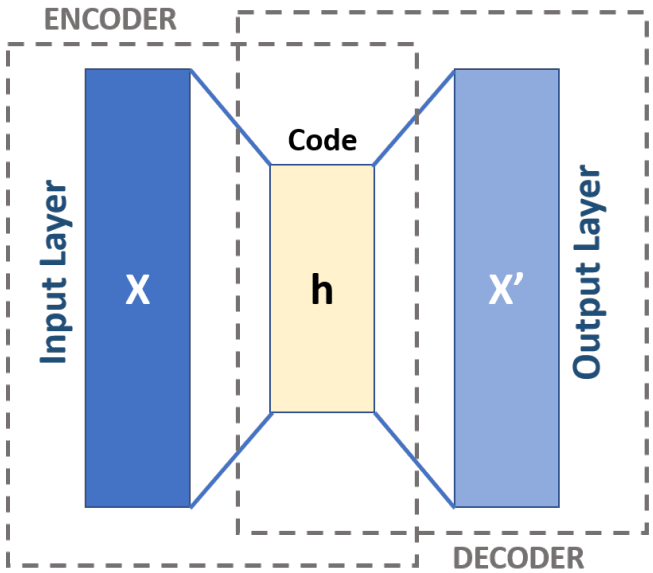
Definition

[1]**Sequntial recommendations** In the setting of sequential recommendation, let $\mathcal{U}$ and $\mathcal{I}$ represent the set of users and items, respectively. For each user $u \in \mathcal{U}$, its action sequence is denoted as $S_u = (i_1; i_2; \ldots; i_k)$, where $i \in \mathcal{I}$, $T_u = (t_1; t_2; \ldots; t_k)$ is the corresponding timestamp sequence of S_u. The set of all S_u is denoted as S. The object of sequential recommendation is to predict the next item of S_u employing sequence information before time t_k and t_k. In general, sequential recommendation task limits the maximum length of S_u to n. When k is greater than n, taking the most recent n items $(i_{k-n}; i_{k-n+1}; \ldots; i_k)$ to make predictions. Each user and item can be converted into low dimensional embedding vector $e_u, e_i \in \mathbb{R}^d$, respectively, where $u \in \mathcal{U}$ and $i \in \mathcal{I}$, d is the dimension of embedding space. We use the $E_{\mathcal{U}} \in \mathbb{R}^{|\mathcal{U}| \times d}$ and $E_{\mathcal{I}} \in \mathbb{R}^{|\mathcal{I}| \times d}$ representing the user embedding and item embedding matrix, respectively.

Example models:

- Matrix Factorization (NMF, SVD, FunkSVD, NeuMF, LightFM, ...)
- GNNs (TAGNN, ...)

Encoder-Decoder



Rysunek: Encoder-decoder (autoencoder) schema.

Example models:

- PCA
- word2vec
- VAE

Source https://en.wikipedia.org/wiki/Autoencoder#/media/File:Autoencoder_schema.png

Self-supervised learning

Definition

Self-supervised learning (SSL) is a paradigm in machine learning where a model is trained on a task using the data itself to generate supervisory signals, rather than relying on external labels provided by humans. In the context of neural networks, self-supervised learning aims to leverage inherent structures or relationships within the input data to create meaningful training signals. Self-supervised learning more closely imitates the way humans learn to classify objects.

- contrastive - uses positive and negative examples
- non-contrastive - uses only positive examples

Example models:

- word2vec

Source: https://en.wikipedia.org/wiki/Self-supervised_learning

DGRS [1]

Dynamic Graph Construction

- nodes are stable
- edges have timestamps, thus we will consider quintuples:
  - u - user
  - i - item
  - t - timestamp (timestamp need to be ordered)
  - o_u^i - the position of item i in all items that user u has interracted with
  - o_i^u - the position of user u in all users that have interracted with item i

Graph at time stamp t:

$$G^t = \{(u, i, t^*, o_u^i, o_i^u) : u \in U, i \in V, t^* \leq t\}$$

Dynamic graph:

$$G = \left(G^t\right)_{t \in T}$$

Sub-Graph Sampling

As the graph grows through the time, we use sub-graph sampling to limit its size:

- select only n items user have lastly interacted with
- for those items, we select only users who are in the m-hop neighborhood
- remember selected nodes as anchor nodes to avoid repetitive sampling

We end up with:

$$G_u^m(t_k)$$

Sub-Graph Sampling

Algorithm 1: Sub-graph Sampling Algorithm

Input : Sequence $S^u = (i_1, i_2, \cdots, i_k)$, timestamp sequence $T^u = (t_1, t_2, \cdots, t_k)$, dynamic graph $\mathcal{G}^{t_k}$, and the order of sub-graph m.

Output: The m-order sub-graph $\mathcal{G}_u^m(t_k)$.

// Initialization
$\mathcal{U}_m, \mathcal{U}_{temp} \leftarrow \{u\}, \mathcal{I}_m, \mathcal{I}_{temp} \leftarrow \{i_1, \cdots, i_k\}, j = 0$
// Node sampling
while $j \leq m$ **do**
  for $i \in \mathcal{I}_{temp}$ **do**
    $\mathcal{U}_{temp} \leftarrow \mathcal{U}_{temp} \cup \mathcal{N}_i$
  $\mathcal{U}_{temp} \leftarrow \mathcal{U}_{temp} \setminus \mathcal{U}_m$
  $\mathcal{U}_m \leftarrow \mathcal{U}_m \cup \mathcal{U}_{temp}$
  if $\mathcal{U}_{temp} = \emptyset$ **then**
    Break
  for $u \in \mathcal{U}_{temp}$ **do**
    $\mathcal{I}_{temp} \leftarrow \mathcal{I}_{temp} \cup \mathcal{N}_u$
  $\mathcal{I}_{temp} \leftarrow \mathcal{I}_{temp} \setminus \mathcal{I}_m$
  $\mathcal{I}_m \leftarrow \mathcal{I}_m \cup \mathcal{I}_{temp}$
  if $\mathcal{I}_{temp} = \emptyset$ **then**
    Break
  $j = j + 1$
// Sub-graph generation
$\mathcal{G}_u^m(t_k) = (\mathcal{U}_m, \mathcal{I}_m), \mathcal{U}_m, \mathcal{I}_m \in \mathcal{G}^{t_k}$

Rysunek: Sub-graph sampling algorithm.

Dynamic Graph Recommendation Networks

- Message passing
  Intuition behind passing the messages in long and in short term:
  - from item to user
    long- and short-term preferences
  - from user to item
    long- and short-term character
- Aggregation

DGRN - message passing - long-term

- GCN:

$$h_u^L = \frac{1}{|\mathcal{N}_u|} \sum_{i \in \mathcal{N}_u} W_1^{(l-1)} h_i^{(l-1)}$$

$$h_i^L = \frac{1}{|\mathcal{N}_i|} \sum_{i \in \mathcal{N}_i} W_2^{(l-1)} h_u^{(l-1)}$$

- GRU

$$h_u^L = GRU_U^{(l)}(h_{i_1}^{(l-1)}, \ldots, h_{i_{|\mathcal{N}_i|}}^{(l-1)})$$

$$h_i^L = GRU_I^{(l)}(h_{u_1}^{(l-1)}, \ldots, h_{u_{|\mathcal{N}_u|}}^{(l-1)})$$

DGRN - message passing - long-term

- Dynamic Graph Attntion Mechanism (DAT) for users:
  - relative order of item:
$$r_u^i = |\mathcal{N}_u| - o_u^i$$
  - relative order embedding $p_r^K \in \mathbb{R}^d$
  - importance weight:
$$e_{ui} = \frac{\left(W_2^{(l-1)} h_u^{(l-1)}\right)^T \left(W_1^{(l-1)} h_i^{(l-1)} + p_{r_u^i}^K\right)}{\sqrt{d}}$$
$$\alpha_{ui} = \text{softmax}(e_{ui})$$
  - hidden state:
$$h_u^L = \sum_{i \in \mathcal{N}_i} \alpha_{ui} \left(W_1^{(l-1)} h_i^{(l-1)} + p_{r_u^i}^K\right)$$
  - Similarly for items

DGRN - message passing - short-term

- Attention mechanism:
  - hidden state:

$$h_u^S = \sum_{i \in \mathcal{N}_i} \hat{\alpha}_{ui} h_i^{(l-1)}$$

  - weights:

$$\hat{\alpha}_{ui} = \text{softmax}\left(\frac{\left(W_3^{(l-1)} h_{i_{|\mathcal{N}_i|}}^{(l-1)} \right)^T \left(W_2^{(l-1)} h_i^{(l-1)} \right)}{\sqrt{d}} \right)$$

  - Similarly for items

DGRN - aggregation (node update)

$$h_u^{(l)} = \tanh(W_3^{(l)}[h_u^L \| h_u^S \| h_u^{(l-1)}])$$
$$h_i^{(l)} = \tanh(W_4^{(l)}[h_i^L \| h_i^S \| h_i^{(l-1)}])$$

where:

- $h_\cdot^L$ is the long-term character/preference
- $h_\cdot^S$ is the short-term character/preference
- $h_\cdot^{(l-1)}$ is the information from previous layer

Recommendation and Optimization

- final embedding

$$h_u = h_u^{(1)} \| h_u^{(2)} \dots \| h_u^{(L)}$$

- score vector s_u

$$s_{ui} = h_u^T W_p e_i$$

- normalized scores

$$\hat{y}_u = \text{softmax}(s_u)$$

- loss

$$\mathcal{L} = -\sum_{S} \sum_{i=1}^{|I|} y_{ui} \log(\hat{y}_{ui}) + (1 - y_{ui}) \log(1 - \hat{y}_{ui}) + \lambda \|\Theta\|_2$$

DGSR framework

Algorithm 2: The DGSR framework (forward propagation)

Input : $S^u = (i_1, i_2, \cdots, i_k)$, timestamp sequence
$T^u = (t_1, t_2, \cdots, t_k)$, all sequences of users, and DGRN layer number L.
Output: The next item i_{k+1} of S^u.

// Dynamic Graph Construction
Convert all user sequences into a dynamic graph $\mathcal{G}$
// The sub-graph of generation for S^u
Run the **Algorithm 1** to generate $\mathcal{G}_u^m(t_k)$ from $\mathcal{G}^{t_k}$
// The initialization of node representation
$\mathbf{h}_u^{(0)} \leftarrow \mathbf{e}_u, \mathbf{h}_i^{(0)} \leftarrow \mathbf{e}_i, \forall u, i \in \mathcal{G}_u^m(t_k)$
// The updte of user and item by DGRN
for $l \in [1 : L]$ **do**
  $\mathbf{h}_u^{(l)}, \mathbf{h}_i^{(l)} \leftarrow \text{DGRN}(\mathbf{h}_u^{(l-1)}, \mathbf{h}_i^{(l-1)}, \mathcal{G}_u^m(t_k))$:
  $\mathbf{h}_u^{(L)}, \mathbf{h}_i^{(L)} \leftarrow$ Long-term Information Encoding
  $\mathbf{h}_u^{(S)}, \mathbf{h}_i^{(S)} \leftarrow$ Short-term Information Encoding
  $\mathbf{h}_u^{(l)} \leftarrow \tanh\left(\mathbf{W_3}^{(l)}\left[\mathbf{h}_u^L \parallel \mathbf{h}_u^S \parallel \mathbf{h}_u^{(l-1)}\right]\right)$
  $\mathbf{h}_i^{(l)} \leftarrow \tanh\left(\mathbf{W_4}^{(l)}\left[\mathbf{h}_i^L \parallel \mathbf{h}_i^S \parallel \mathbf{h}_i^{(l-1)}\right]\right)$
// The prediction of next item.
$\mathbf{h_u} = \mathbf{h}_u^{(0)} \parallel \mathbf{h}_u^{(1)}, \cdots, \parallel \mathbf{h}_u^{(L)}$
Next item $\leftarrow \text{argmax}_{i \in \mathcal{V}}(\mathbf{h_u}^{\mathrm{T}} \mathbf{W_P} \mathbf{e}_i)$

Evaluation

TABLE 2: Performance of DGSR and compared methods in terms of Hit@10 and NDCG@10. The best results is boldfaced. The underlined numbers is the second best results. "Gain" means the improvement over the best compared methods.

Datasets	Metric	BPR-MF	FPMC	GRU4Rec+	Caser	SASRec	SR-GNN	HGN	TiSASRec	HyperRec	DGSR	Gain
Beauty	NDCG@10	21.83	28.91	26.42	25.47	32.19	32.33	32.47	30.45	23.26	**35.90**	10.56%
	Hit@10	37.75	43.10	43.98	42.64	48.54	48.62	48.63	46.87	34.71	**52.40**	7.75%
Games	NDCG@10	28.75	46.80	45.64	45.93	53.60	53.25	49.34	50.19	48.96	**55.70**	3.92%
	Hit@10	37.75	68.02	67.15	68.83	73.98	73.49	71.42	71.85	71.24	**75.57**	2.15%
CDs	NDCG@10	36.26	33.55	44.52	45.85	49.23	48.95	49.34	48.97	47.16	**51.22**	3.81%
	Hit@10	56.27	51.22	67.84	68.65	71.32	69.63	71.42	71.00	71.02	**72.43**	1.41%

Ablation Study

TABLE 3: Performance of compared with different model variants in terms of NGCD@10 and Hit@10 ("–" indicates DGSR does not consider the setting of this part).

Variants	Ablation		Beauty		Games		CDs	
	Long-term	Short-term	NDCG@10	Hit@10	NDCG@10	Hit@10	NDCG@10	Hit@10
DGSR-G	GCN	–	33.75	49.94	53.44	73.23	48.66	70.43
DGSR-R	RNN	–	34.81	50.90	54.70	74.73	49.57	71.22
DGSR-D	DAT	–	35.25	51.36	55.12	74.83	49.66	71.26
DGSR-L	–	Last	30.87	46.13	52.43	72.18	46.23	67.38
DGSR-A	–	ATT	34.76	51.00	54.30	74.32	48.78	70.09
DGSR-GL	GCN	Last	35.24	51.18	54.76	74.58	49.62	70.76
DGSR-RL	RNN	Last	35.47	51.68	54.86	74.84	50.26	71.24
DGSR-DL	DAT	Last	35.62	51.92	55.53	75.07	50.72	72.06
DGSR-GA	GCN	ATT	35.00	51.05	54.97	74.78	50.05	71.46
DGSR-RA	RNN	ATT	35.17	51.46	55.02	74.88	51.19	**72.55**
DGSR-DA	DAT	ATT	**35.90**	**52.40**	**55.70**	**75.57**	**51.22**	72.43

DyG2Vec [2]

DyG2Vec - Idea

- We will use the encoder-decoder idea (encoder to provide representation, decoder to gather specific knowledge)

$$\text{Encoding: } H = g_\theta(G), \quad \text{Decoding: } z = d_\gamma(H, \bar{e}),$$

- we will consider temporal graphs $G_{i,j}$ containing only interactions that happend between time i and time j.
- Idea: you can specify the downstream task or do some self-supervised pre-training.
- Pre-training: first learn $f^{SSL} = (g_\theta, d_\psi)$ with self-supervision, then we rain only the task specific decoder d_γ.

DyG2Vec - Encoder

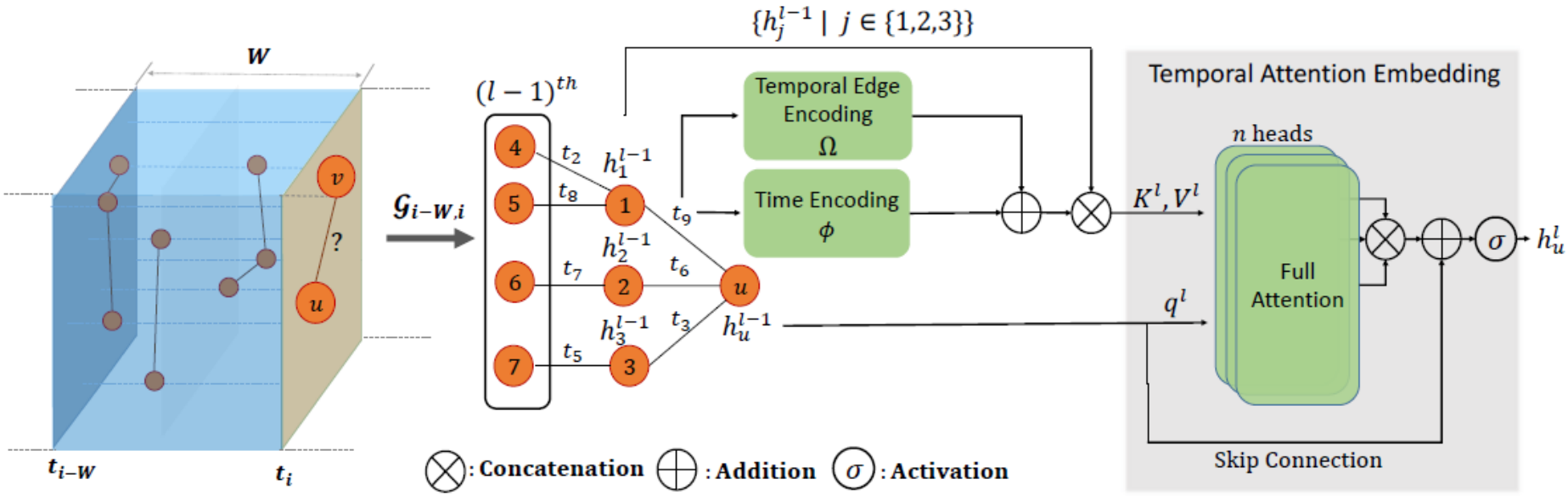


Figure 1: Using DyG2Vec window framework to encode the target node u. Every slice of the dynamic graph $\mathcal{G}$ contains edges that arrived at the same continuous timestamp. The blue interval represents the history graph $\mathcal{G}_{i-W,i}$ that is encoded to make a prediction on the target edge (u, v). Note that both u and v share the same sampled history graph. For simplicity, we omit edge features m_p from the attention encoder.

Temporal Edge Encoding & Time Encoding

- Temporal Edge Encoding

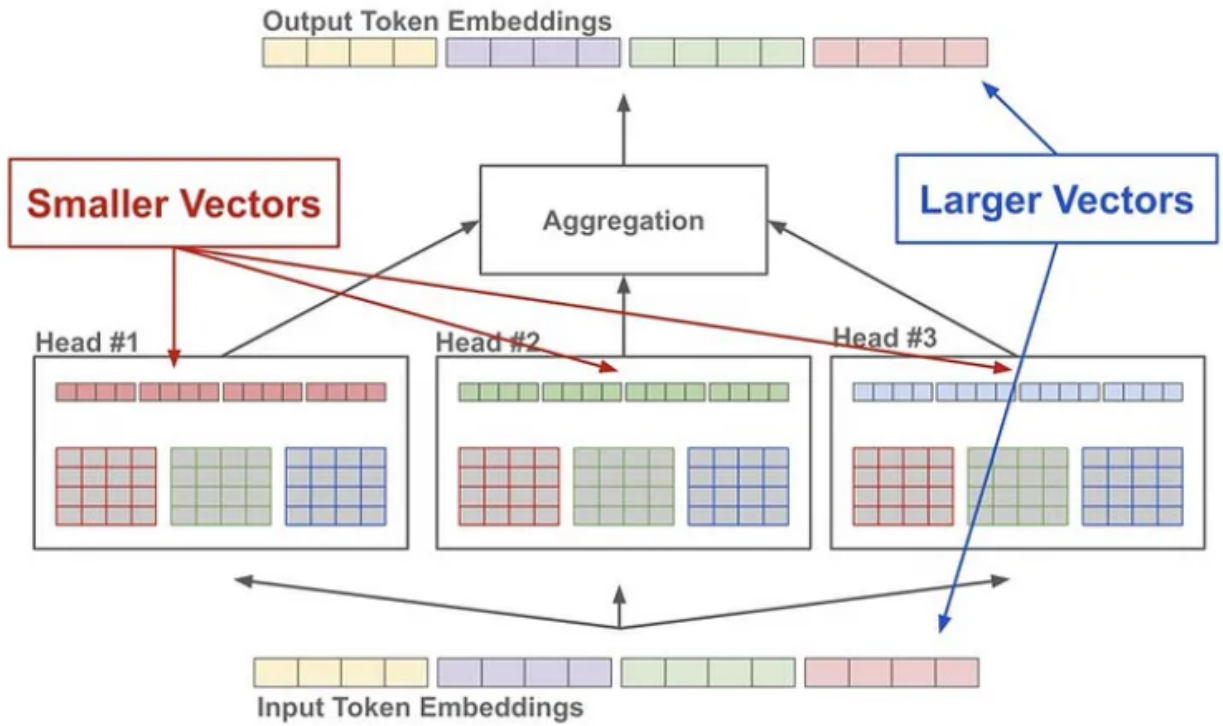
$$\Omega_p(t_p) = W_2[z_p(t_p) \| c_p(t_p)],$$

where $z_p(t_p) \in \mathbb{R}^2$ is temporal degree centrality (current degrees of nodes u_p, v_p), $c_p(t_p)$ - common neighbors (common 1-hop neighbors between u_p and v_p at time t_p).

- Time Encoding

$$\phi(t) = [\cos \omega_1 t, \ldots, \cos \omega_{D^1} t]$$

Multi-head Attention



Rysunek: Multi-head Attention schema (weighting attention in multiple heads in paralel).

Source: https://medium.com/@akash.kesrwani99/multi-head-self-attention-short-understanding-e90a34866730

Temporal Attention Embedding

- We will consider only a subset of the neighborhood (sample n neighbors unifromly at random), where
  - p is the index of first/earlies sampled item
  - k is the index of last/latest sampled item
- The representations in consequtive layers are transformed as:

$$h_i^(l) = W_1 h_i^{(l-1)} + MHA(h_i^{(l-1)}, K^l, V^l),$$

where:

- $K^l = V^l = [\Phi_p^{(l-1)}(t_p), \ldots, \Phi_k^{(l-1)}{}_p(t_k)]$
- $MHA(.)$ is a multi-head dot-product attention layer
- $\Phi_p^{(l-1)}(t_p) = [h_{u_p}^{(l-1)} \| f_p(t_p) \| m_p]$
- $f_p(t_p) = \phi(\bar{t}_i - t_p) + \Omega_p(t_p)$
- $\bar{t}_i = \max\{t_l | e_l$ was samled$\}$

Self-supervised pretraining for Dynamic Graphs

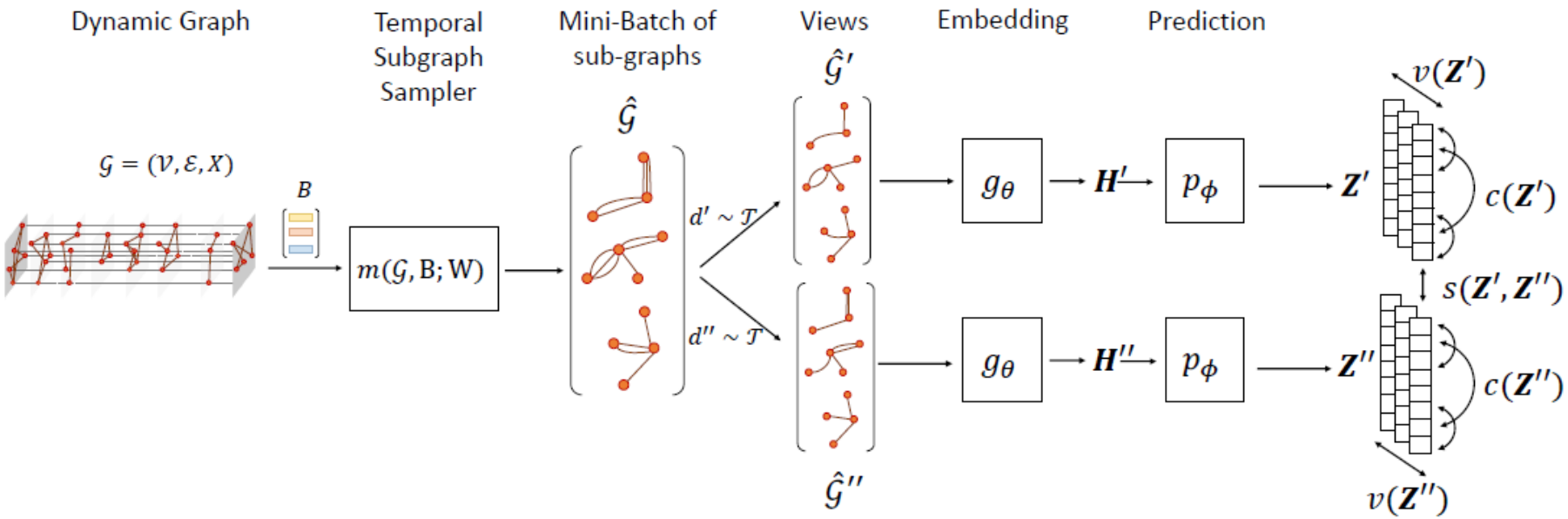


Figure 2: The joint embedding architecture for the non-contrastive SSL Framework. Each slice of the input dynamic graph contains edges arriving at the same continuous timestamp. B is a batch of intervals of size W. $\hat{\mathcal{G}}$ is a batch of the corresponding input graphs of each interval.

Self-supervised pretraining for Dynamic Graphs

- generate a set of intervals by dividing the entire time-span into $M = \lceil E/S \rceil - 1$ intervals withs tride S and length W
- select a minibatch $B \subset I$
- sample input graphs with temporal subgraph sampler
- generate two views of a subgrapgh (with edge dropout and edge feature masking) G', G''
- Embedd the views with Attention-based Message-Passing Encoder H', H''
- provide predition (MLP) - the final reprezentation $Z = p_\phi(H)$
- Optimize loss:

$$\mathcal{L}^{SSL} = s(Z', Z'') + \mu[v(Z') + v(Z'')] + \nu[c(Z') + c(Z'')]$$

where:

- λ, μ, ν control the emphasis of each regularization term
- s is the similarity (of two views on the same graph)
- v is the variance term (to prevent collapse problem)
- c is the content term (for maximizing representation informativness)

Evaluation

Table 2: Future link Prediction Performance in AP (Mean ± Std). **Bold** font and ul font represent first- and second-best performance respectively. DyG2Vec is trained end-to-end with no pre-training.

Setting	Model	Wikipedia	Reddit	MOOC	LastFM	Enron	UCI	SocialEvol.
Transductive	JODIE	0.956 ± 0.002	0.979 ± 0.001	0.797 ± 0.01	0.691 ± 0.010	0.785 ± 0.020	0.869 ± 0.010	0.847 ± 0.014
	DyRep	0.955 ± 0.004	0.981 ± 1e-4	0.840 ± 0.004	0.683 ± 0.033	0.795 ± 0.042	0.524 ± 0.076	0.885 ± 0.004
	TGAT	0.968 ± 0.001	0.986 ± 3e-4	0.793 ± 0.006	0.633 ± 0.002	0.637 ± 0.002	0.835 ± 0.003	0.631 ± 0.001
	TGN	0.986 ± 0.001	0.985 ± 0.001	0.911 ± 0.010	0.743 ± 0.030	0.866 ± 0.006	0.843 ± 0.090	0.966 ± 0.001
	CaW	0.976 ± 0.007	0.988 ± 2e-4	0.940 ± 0.014	0.903 ± 1e-4	0.970 ± 0.001	0.939 ± 0.008	0.947 ± 1e-4
	NAT	0.987 ± 0.001	0.991 ± 0.001	0.874 ± 0.004	0.859 ± 1e-4	0.924 ± 0.001	0.944 ± 0.002	0.944 ± 0.010
	DyG2Vec	**0.995 ± 0.003**	**0.996 ± 2e-4**	**0.980 ± 0.002**	**0.960 ± 1e-4**	**0.991 ± 0.001**	**0.988 ± 0.007**	**0.987 ± 2e-4**
Inductive	JODIE	0.891 ± 0.014	0.865 ± 0.021	0.707 ± 0.029	0.865 ± 0.03	0.747 ± 0.041	0.753 ± 0.011	0.791 ± 0.031
	DyRep	0.890 ± 0.002	0.921 ± 0.003	0.723 ± 0.009	0.869 ± 0.015	0.666 ± 0.059	0.437 ± 0.021	0.904 ± 3e-4
	TGAT	0.954 ± 0.001	0.979 ± 0.001	0.805 ± 0.006	0.644 ± 0.002	0.693 ± 0.004	0.820 ± 0.005	0.632 ± 0.005
	TGN	0.974 ± 0.001	0.954 ± 0.002	0.855 ± 0.014	0.789 ± 0.050	0.746 ± 0.013	0.791 ± 0.057	0.904 ± 0.023
	CaW	0.977 ± 0.006	0.984 ± 2e-4	0.933 ± 0.014	0.890 ± 0.001	0.962 ± 0.001	0.931 ± 0.002	0.950 ± 1e-4
	NAT	0.986 ± 0.001	0.986 ± 0.002	0.832 ± 1e-4	0.878 ± 0.003	0.949 ± 0.010	0.926 ± 0.010	0.952 ± 0.006
	DyG2Vec	**0.992 ± 0.001**	**0.991 ± 0.002**	**0.938 ± 0.010**	**0.979 ± 0.006**	**0.987 ± 0.004**	**0.976 ± 0.002**	**0.978 ± 0.010**

Evaluation

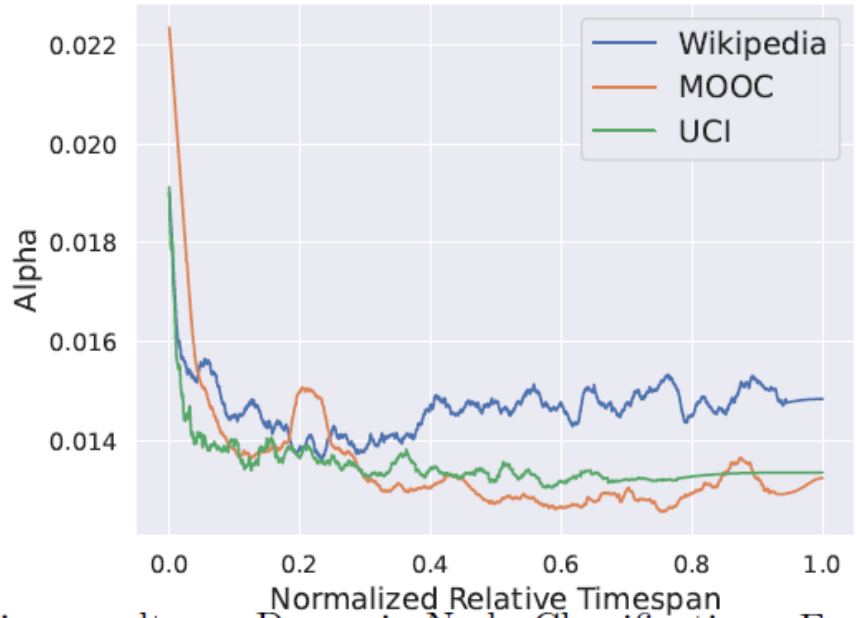


Figure 4: First figure plots Semi-Supervised Learning results on Dynamic Node Classification. For each setting, DyG2Vec was trained on a varying random portion of the training data. Second figure plots the Average Attention Weight versus Relative Timespan for DyG2Vec trained with $W = 64K$. The relative timespan is normalized with the maximum timespan across all interactions. A higher timespan means a farther interaction.

GNN4TS [3]

What dependencies can occur in time and space?

- new edges (for example new connections on LinkedIn)
- different features (for example being on sale in recommender systems)

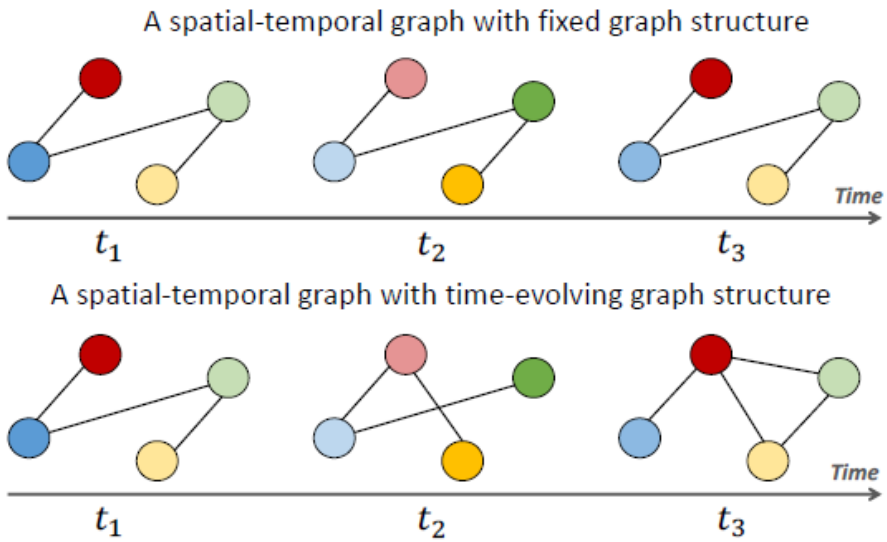


Fig. 2: Examples of spatial-temporal graphs.

Possible GNN4RS tasks [3]

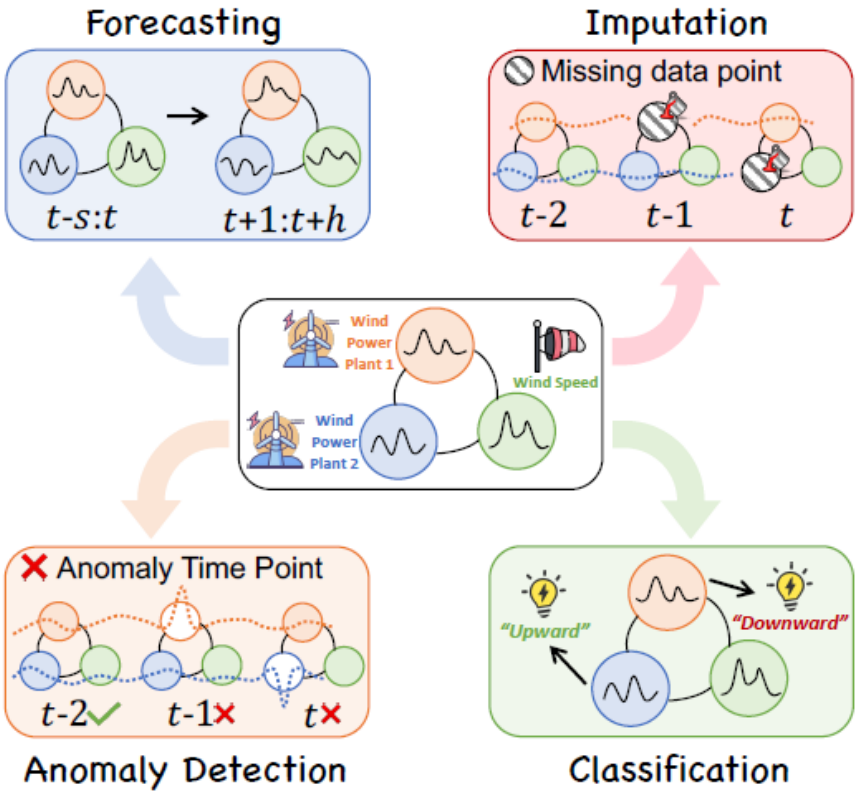
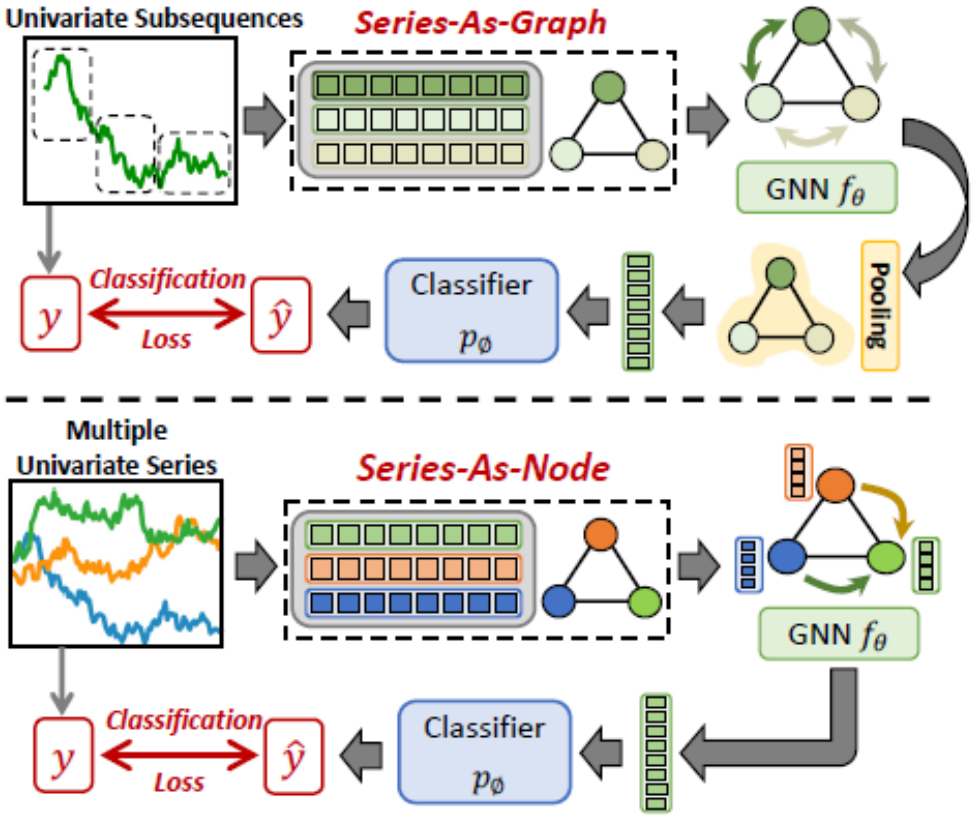


Fig. 1: Graph neural networks for time series analysis (GNN4TS). In this example of wind farm, different analytical tasks can be categorized into time series forecasting, classification, anomaly detection, and imputation.

Detailed task taxonomy

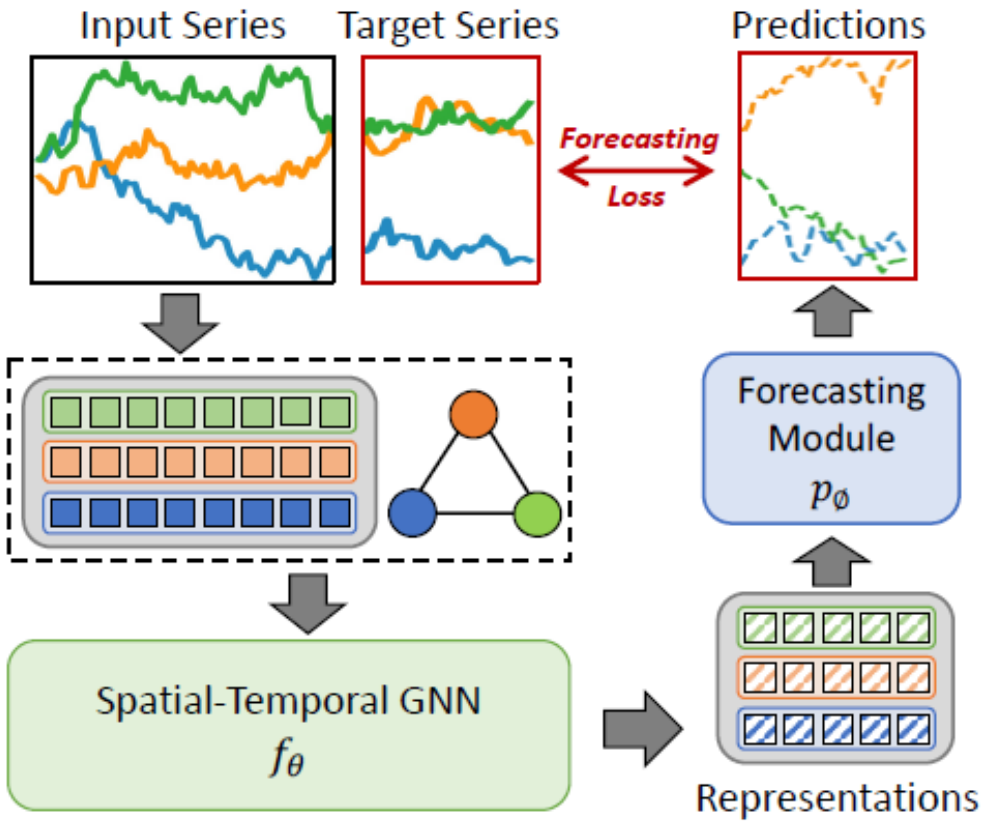
Fig. 3: Task-oriented taxonomy of graph neural networks for time series analysis in the existing literature.

Classification



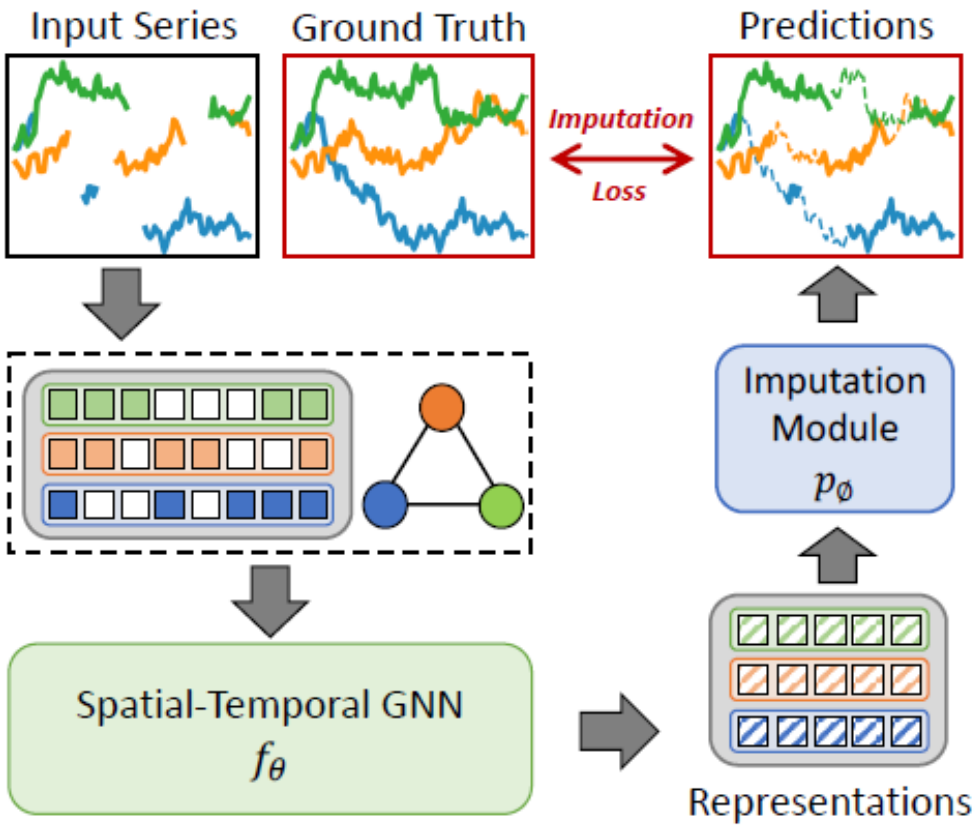
(d) Graph neural networks for time series classification: Formulates green series classification as a graph (top) or node (bottom) classification task.

Forecasting



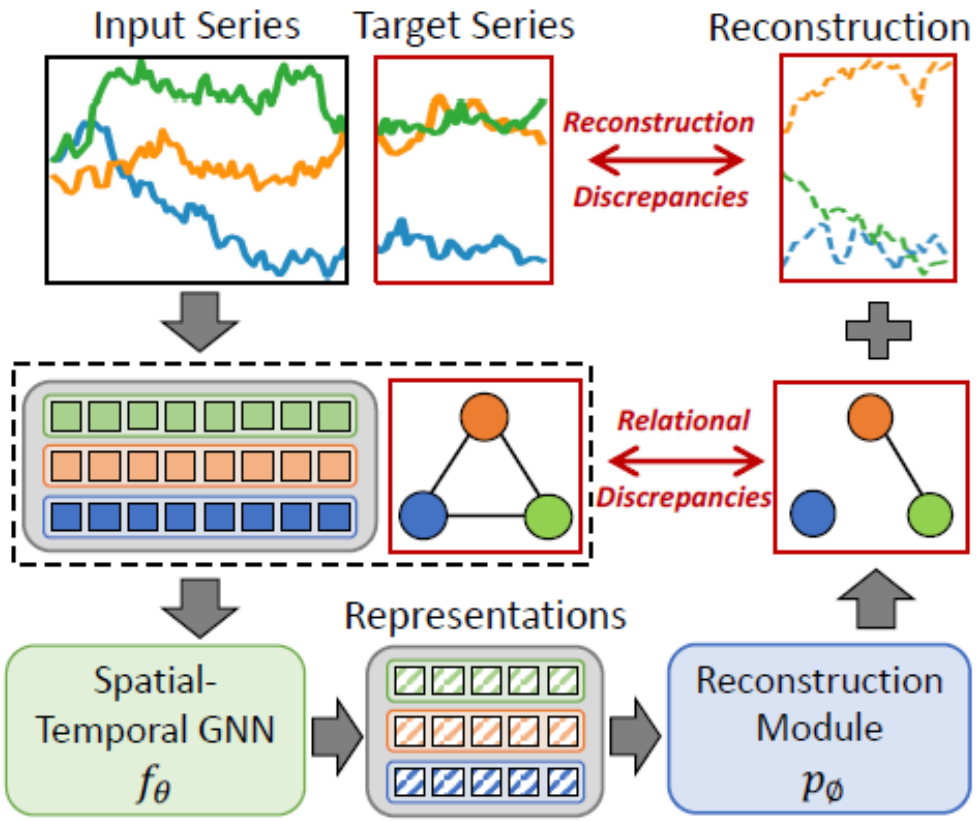
(a) Graph neural networks for time series forecasting.

Imputation



(c) Graph neural networks for time series imputation.

Anomaly Detection



(b) Graph neural networks for time series anomaly detection.

Applications

- Smart transportation (traffic prediction, flight delay prediction, estimation of missing or incomplete traffic data, autonomous driving)
- On-Demand Services (ride-hailing services, bike-sharing services, energy demand, temporal trends and spatial dependencies in tourism data)
- Environment & Sustainable Energy (wind speed and power prediction, solar irradiance, air pollution)
- Internet-of-Things (IoTs)
- Healthcare (patient monitoring sensors, brain functional connectivity, neuroimaging data, ambulance demand, predict health equipment useful life, epidemic disease outbreaks)
- Fraud Detection
- other

Final remarks

Take home messages

- Modelling time series with GNNs allow to capture both time component and spacial information/relations.
- We have tools to capture the non-stationary scenario.
- GNNs are broadly applicated.

Bibliography

[1] Mengqi Zhang, Shu Wu, Xueli Yu, and Liang Wang. Dynamic graph neural networks for sequential recommendation. *CoRR*, abs/2104.07368, 2021.

[2] Mohammad Ali Alomrani, Mahdi Biparva, Yingxue Zhang, and Mark Coates. Dyg2vec: Representation learning for dynamic graphs with self-supervision, 2023.

[3] Ming Jin, Huan Yee Koh, Qingsong Wen, Daniele Zambon, Cesare Alippi, Geoffrey I Webb, Irwin King, and Shirui Pan. A survey on graph neural networks for time series: Forecasting, classification, imputation, and anomaly detection. *arXiv preprint arXiv:2307.03759*, 2023.

GNN4TS

GNNs for Time Series with focus on Dynamic GNNs with application in RS

Klaudia Balcer

Computational Intelligence Research Group, Institute of Computer Science

28 lutego 2024

Uniwersytet Wrocławski