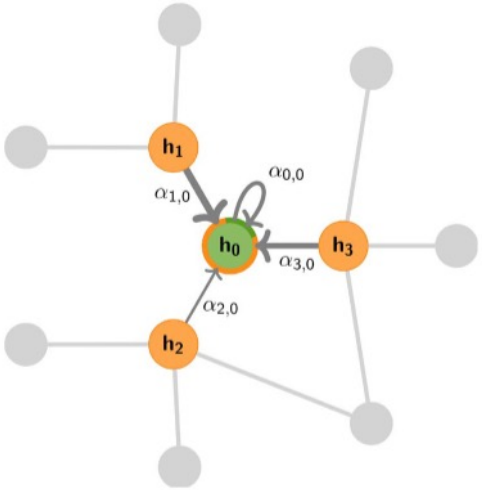
Introduction to Graph Neural Networks



Flavio Morelli

March 15, 2022

About me

- Machine Learning Intern @ Bayer Pharmaceuticals
- M.Sc. in Statistics @ Humboldt University Berlin
- Co-organizer @ Berlin Bayesians

Material / Contact

- Slides can be found on my website: flaviomorelli.com/talks
- Twitter: @mexiamorelli
- Linkedin: linkedin.com/in/mexiamorelli

Main idea

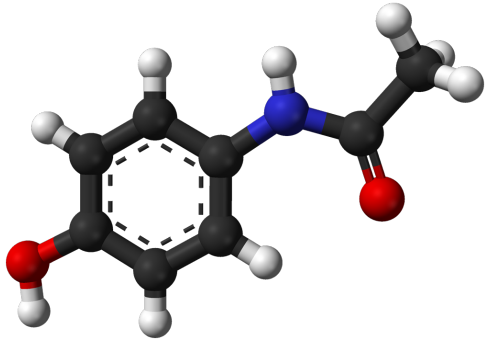
- In statistics, we usually use data in **tabular** shape (rows=observations, columns=variables, order doesn't matter)

- However, there are many types of data that do not follow this **structure**: graphs, images, manifolds

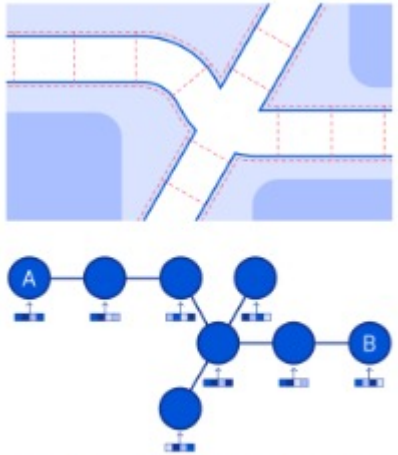
- Ignoring the **geometric information** in those types of data might lead to suboptimal performance

- How can we find a **good representation** of a graph that we can use for downstream tasks (i.e., regression, classification, etc.)?

Graph examples



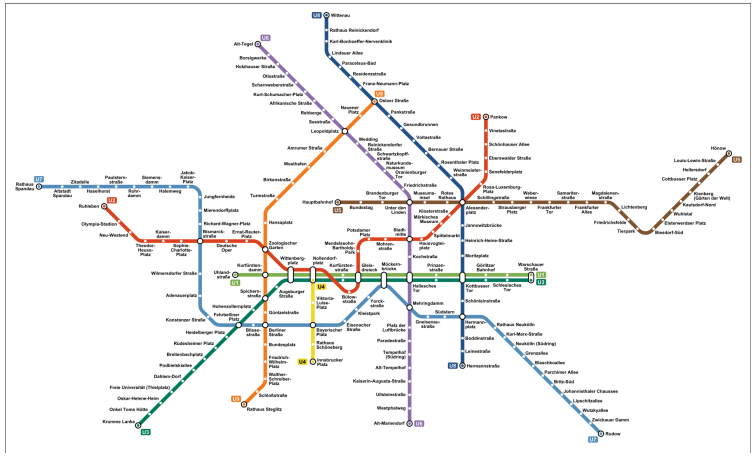
Paracetamol



A road network (top) with its corresponding graph representation (bottom).

Bronstein et al. (2021)

Social networks



BVG

Contents

1. Notation
2. Basics of GNNs
3. Application

Contents

Graph notation

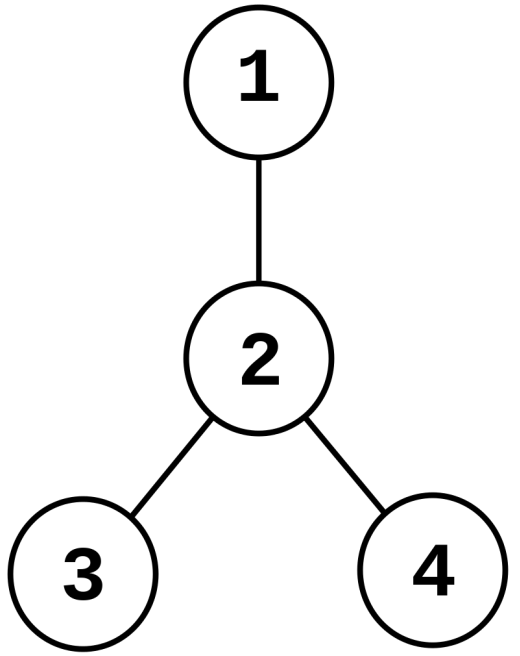
Graph: $G = (V, E)$

Nodes: $V = \{1, 2, 3, 4\}$

Edges: $E = \{(1, 2),\ (2, 1), (2, 3),\ (3, 2), (2, 4), (4, 2)\}$

Degree: # of in and out edges to and from node:
$deg_1 = 2, deg_2 = 6, deg_3 = 2, deg_4 = 2$

Node features: $X_v, v \in V$ vector/matrix of real values

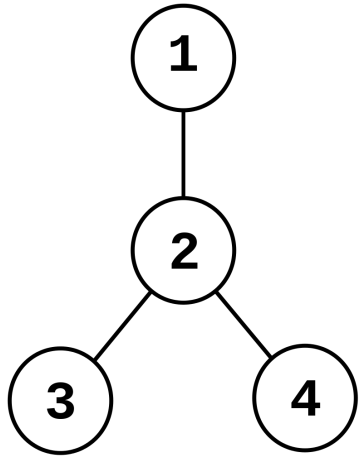


Wikipedia

Adjacency matrices

How to represent a graph?
Adjacency matrix A

	1	2	3	4
1	0	1	0	0
2	1	0	1	1
3	0	1	0	0
4	0	1	0	0



Source: Wikipedia

- Adjacency matrices tend to be **sparse** → memory inefficient
- Adjacency matrices scale **quadratically** with the number of nodes, i.e., $O(|V|^2)$

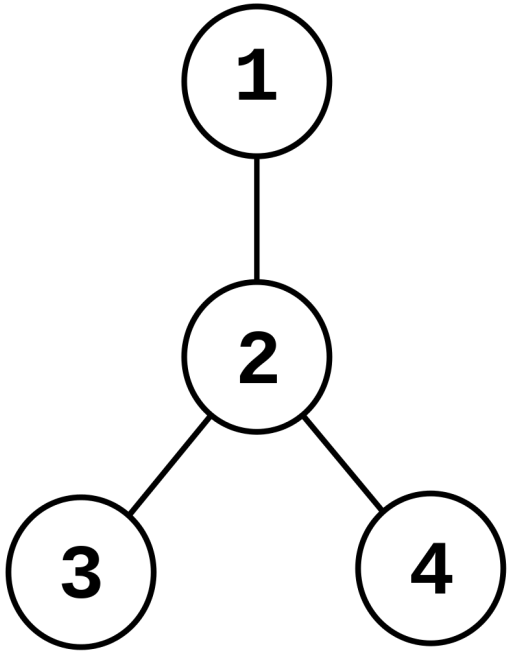
Contents

Types of prediction tasks

1. Node prediction

2. Edge prediction

3. Graph prediction



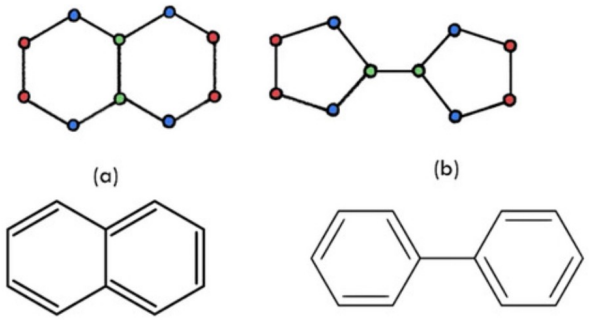
Prediction tasks in detail

1. Node prediction: for each node $v \in V$ with a dependent variable y_v, find a representation h_v such that $\hat{y}_v = f(h_v)$

2. Edge prediction: not discussed here

3. Graph prediction: $h_G = \mathrm{READOUT}\left(\left\{h_v^{(K)} \mid v \in G\right\}\right)$

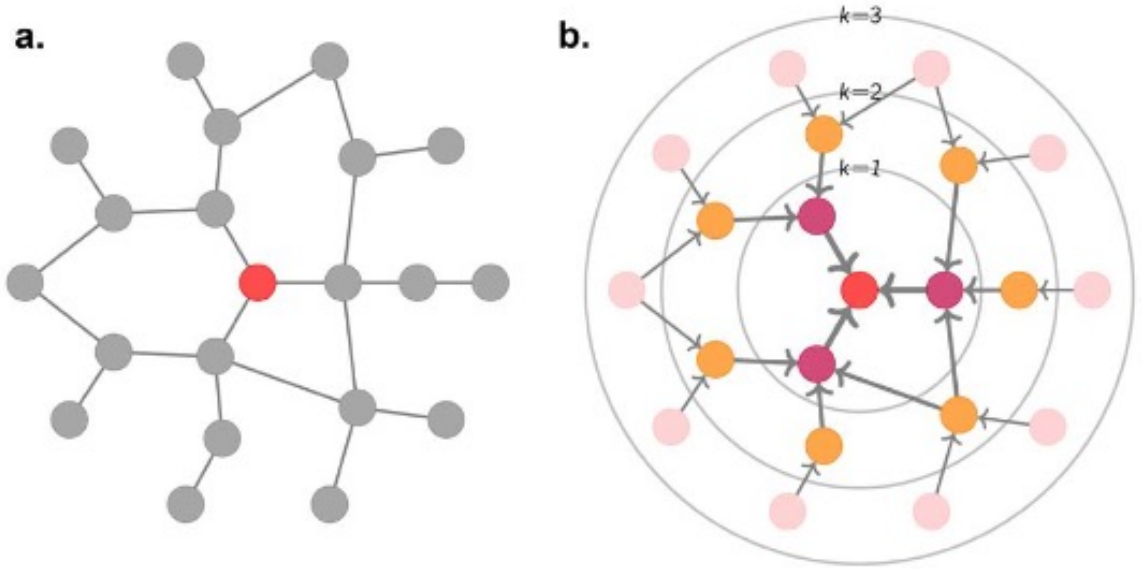
What is a good representation?



Ying et al. (2021)

- **Intuition**: different graphs should have different representations
- The Weisfeiler-Lehmann test is commonly use to detect graph isomorphism
- Ideally, GNNs should be as powerful as the WL test

How do GNNs work?



Gaudelet et al. (2021)

How do GNNs work?

- Usually, GNNs use an **iterative** aggregation strategy of its neighbors' representations

- After *k* iterations, the representation $h_v^{(k)}$ captures the information of the *k*-hop neighborhood of node *v*.

- Formally,

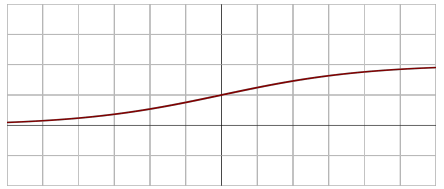
$$a_v^{(k)} = \text{AGGREGATE}^{(k)}\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right), \quad h_v^{(k)} = \text{COMBINE}^{(k)}\left(h_v^{(k-1)}, a_v^{(k)}\right)$$

Xu et al. (2018)

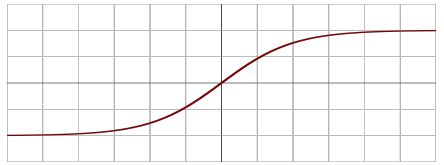
where $h_v^{(0)} = X_v$ and $\mathcal{N}(v)$ is the set of nodes adjacent to *v*

Digression: nonlinearity in DL

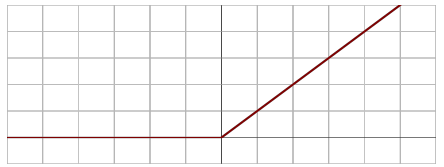
- In the original scale, it might be hard to use simple methods such as linear models.
- Using nonlinear transformations can simplify the learning problem
- Examples of non-linearities:
  - Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$



  - Hyperbolic tangent: $\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$



  - *ReLU*: $ReLU(x) = \max(0, x)$



Common approaches

- GraphSAGE (Hamilton et al. 2017)

$$a_v^{(k)} = \text{MAX}\left(\left\{\text{ReLU}\left(W \cdot h_u^{(k-1)}\right), \forall u \in \mathcal{N}(v)\right\}\right) \qquad h_v^{(k)} = W \cdot \left[h_v^{(k-1)}, a_v^{(k)}\right]$$

- GCN (Kipf & Welling, 2017) integrate COMBINE and AGGREGATE in one step

$$h_v^{(k)} = \text{ReLU}\left(W \cdot \text{MEAN}\left\{h_u^{(k-1)}, \forall u \in \mathcal{N}(v) \cup \{v\}\right\}\right)$$

- GIN (Xu et al., 2018) derive a GNN as powerful as the WL-test

$$h_v^{(k)} = \text{MLP}^{(k)}\left(\left(1 + \epsilon^{(k)}\right) \cdot h_v^{(k-1)} + \sum\nolimits_{u \in \mathcal{N}(v)} h_u^{(k-1)}\right)$$

Contents

Github dataset

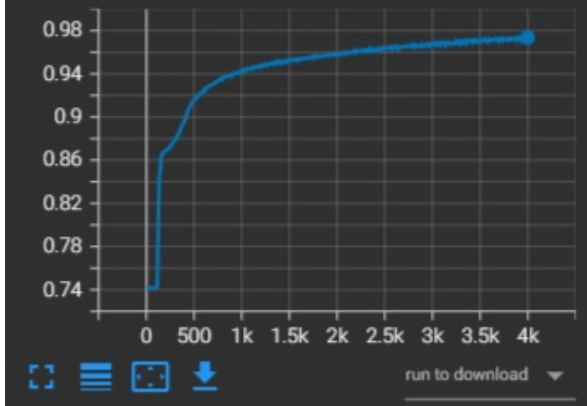
- Social network data (Rozcemberski, 2021)
- Nodes are **developers** (n=37700), who have starred at least 10 repositories on GitHub
- Node **features** (k=128) are location, starred repositories, employer
- **Edges** are mutual follower relations
- Task: identify whether developers are **web** or **machine learning** developers

```python
import torch_geometric as pyg
from torch_geometric.nn import GCNConv
from torch import nn
import torch

class GNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = GCNConv(128, 64)
        self.conv2 = GCNConv(64, 32)
        self.linear = nn.Sequential(
            nn.Linear(32, 16),
            nn.ReLU(),
            nn.Linear(16, 8),
            nn.ReLU(),
            nn.Linear(8, 4),
            nn.ReLU(),
            nn.Linear(4, 1),
        )

    def forward(self, data):
        x, edge_index = data.x, data.edge_index

        x = self.conv1(x, edge_index)
        x = torch.relu(x)
        x = self.conv2(x, edge_index)
        x = torch.relu(x)

        x = torch.sigmoid(self.linear(x))

        return x
```

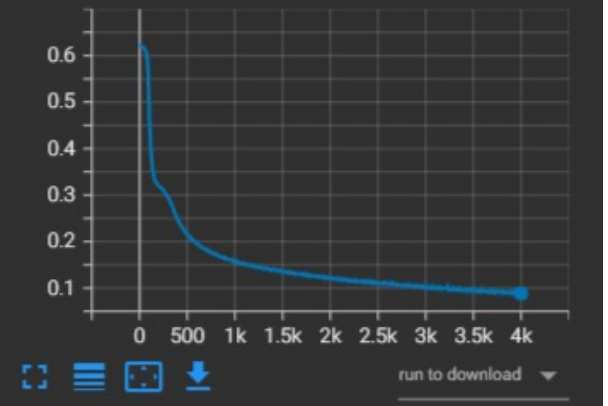
$$h_v^{(k)} = \mathrm{ReLU}\left(W \cdot \mathrm{MEAN}\left\{h_u^{(k-1)}, \forall u \in \mathcal{N}(v) \cup \{v\}\right\}\right)$$

Kipf & Welling (2017)

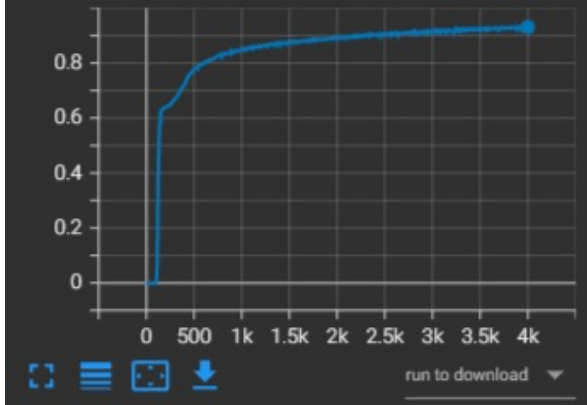
Loss_Train/Accuracy
tag: Loss_Train/Accuracy



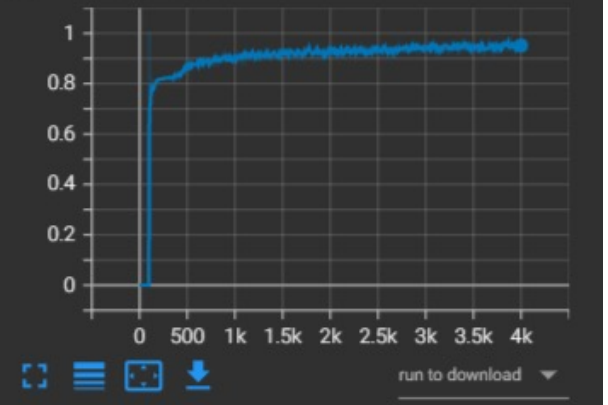
Loss_Train/BCE
tag: Loss_Train/BCE



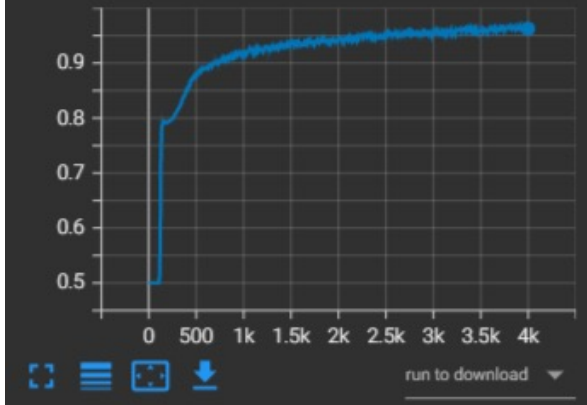
Loss_Train/Matthews Correlation
tag: Loss_Train/Matthews Correlation



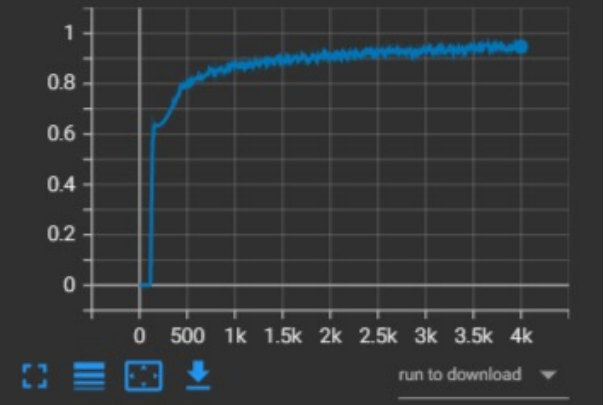
Loss_Train/Precision
tag: Loss_Train/Precision



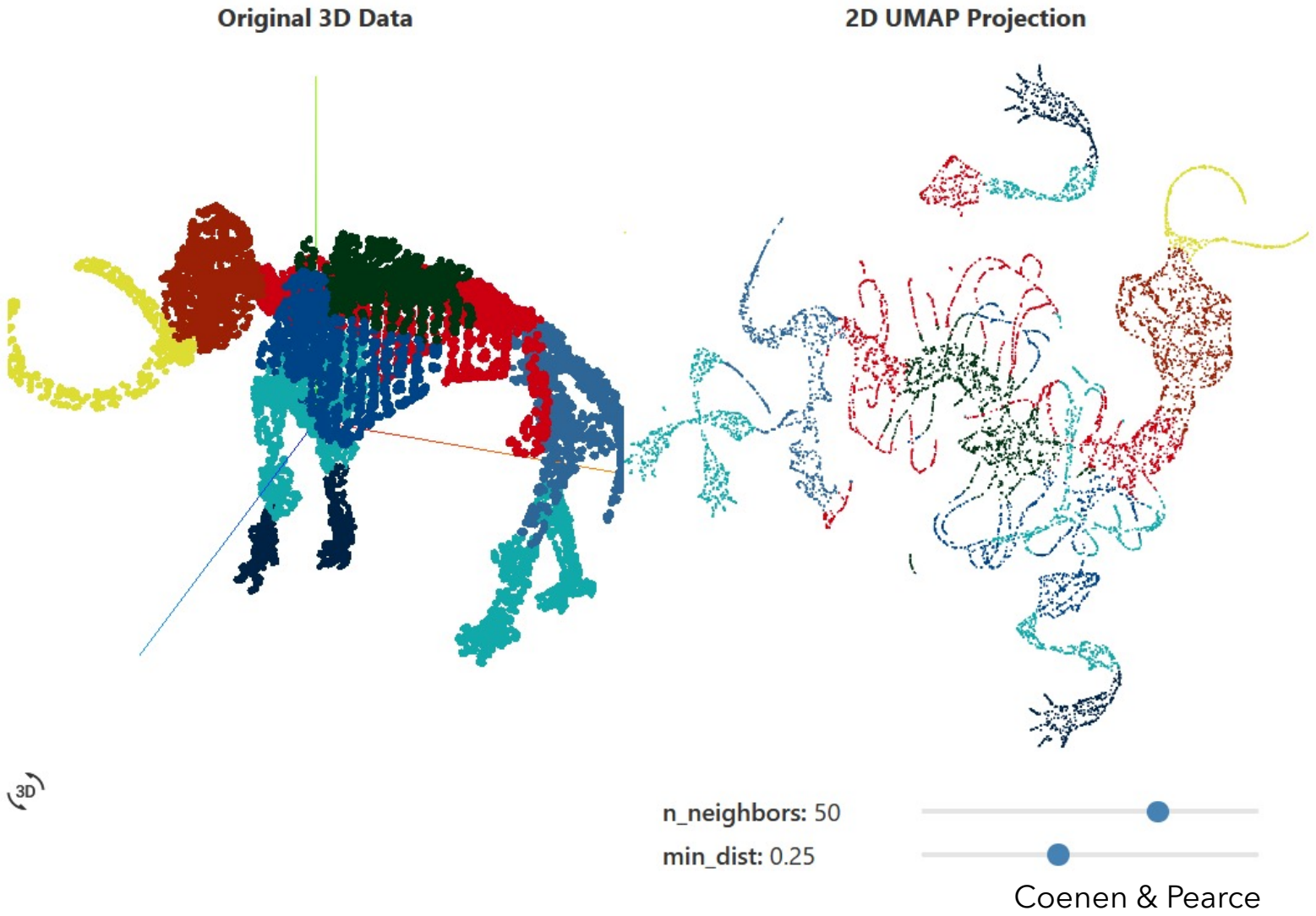
Loss_Train/ROC AUC
tag: Loss_Train/ROC AUC



Loss_Train/Recall
tag: Loss_Train/Recall

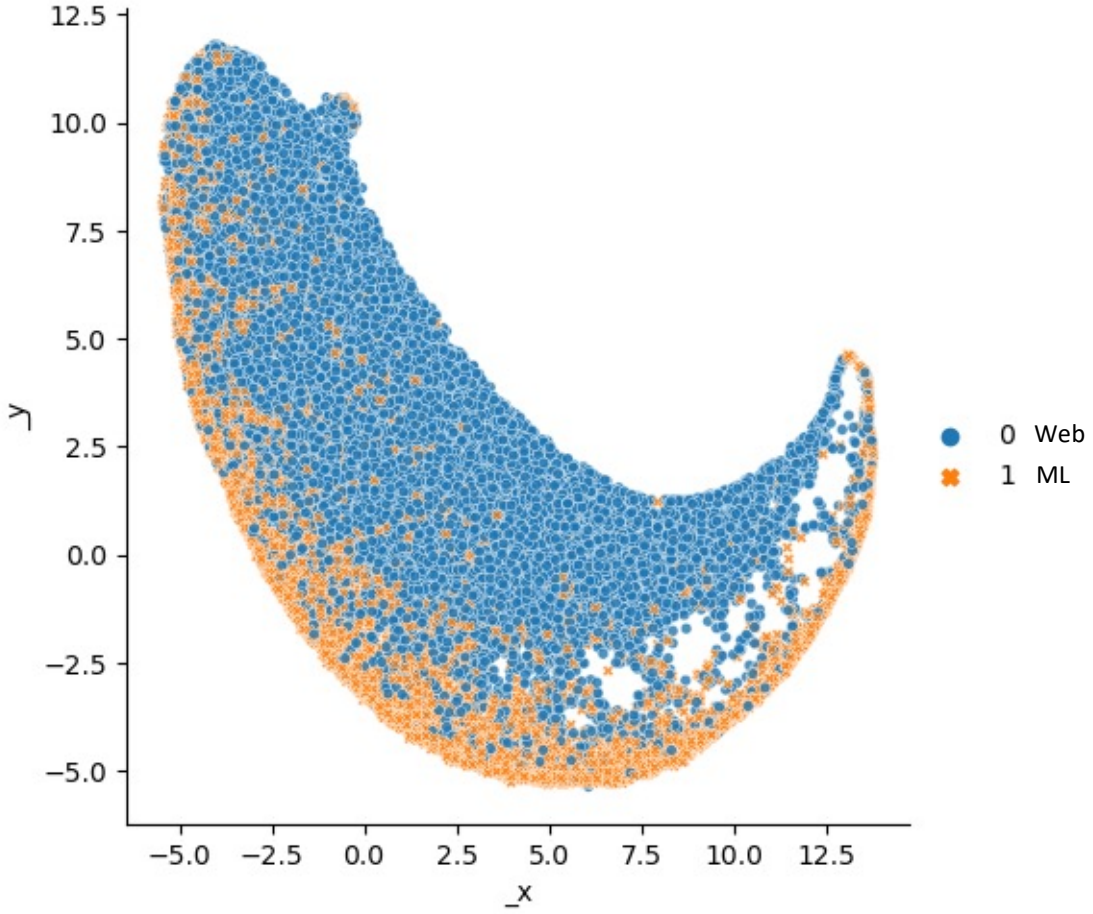


Digression: UMAP



Coenen & Pearce

2D-projection from Graph Convolution



Summary

- GNNs are a powerful tool that enable learning tasks on graphs
- GNNs consider the geometric structure of the graph and therefore needs less observations than other DL approaches
- There are many types of GNNs, and a more abstract presentation can be found in Bronstein et al. 2021
- In the SAE context, GNNs can be applied to non-standard data sources

References

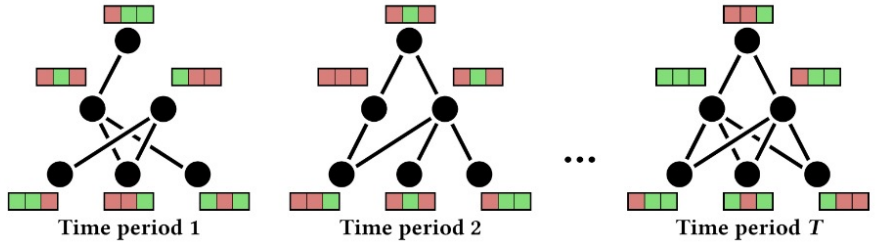
- Bronstein, M. M., Bruna, J., Cohen, T., & Veličković, P. (2021). Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*
- Coenen, A., & Pearce, Adam: Understanding UMAP. https://pair-code.github.io/understanding-umap/ (retrieved 2022/01/27)
- Gaudelet, T., Day, B., Jamasb, A. R., Soman, J., Regep, C., Liu, G., ... & Taylor-King, J. P. (2021). Utilizing graph machine learning within drug discovery and development. *Briefings in bioinformatics*, *22*(6)
- Hamilton, W. L., Ying, R., & Leskovec, J. (2017, December). Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (pp. 1025-1035)
- Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*
- Rozemberczki, B., Allen, C., & Sarkar, R. (2021). Multi-scale attributed node embedding. *Journal of Complex Networks*, *9*(2), cnab014
- Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2018). How powerful are graph neural networks?. *arXiv preprint arXiv:1810.00826*
- Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., ... & Liu, T. Y. (2021). Do Transformers Really Perform Bad for Graph Representation?. *arXiv preprint arXiv:2106.05234*

Temporal GNNs

Contents

Contents

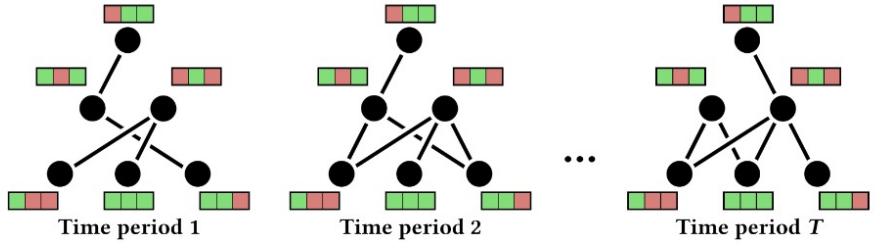
Overview: evolving graphs



(a) Dynamic graph with temporal signal.

Supply chains: companies change, relations with suppliers change

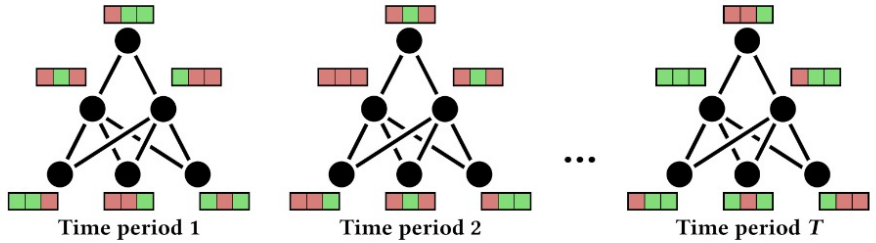
$$\mathcal{D} = \{(\mathcal{G}_1, X_1), \ldots, (\mathcal{G}_T, X_T)\}$$
$$V_t = V, \ \forall t \in \{1, \ldots, T\}$$
$$X_t \in \mathbb{R}^{|V| \times d}$$



(b) Dynamic graph with static signal.

Citation networks: abstract and title stay the same, but citations increase

$$\mathcal{D} = \{(\mathcal{G}_1, X), \ldots, (\mathcal{G}_T, X)\}$$
$$V_t = V, \ \forall t \in \{1, \ldots, T\}$$
$$X \in \mathbb{R}^{|V| \times d}$$



(c) Static graph with temporal signal.

Traffic networks:

it's hard to build a street overnight!

$$\mathcal{D} = \{(\mathcal{G}, X_1), \ldots, (\mathcal{G}, X_T)\}$$
$$X_t \in \mathbb{R}^{|V| \times d}$$

Rozemberczki et al. (2021)

Overview: what kind of time?

Discrete

separate & distinct points in time

usually sampled from continuous signal

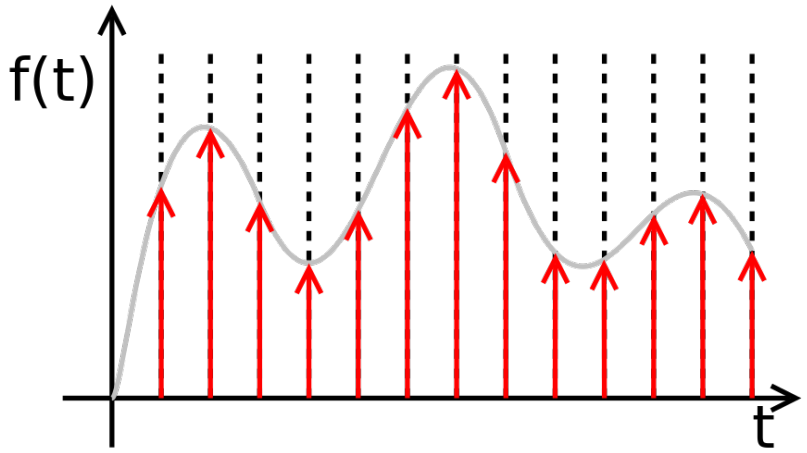
most ML methods in this category

Continuous

no predetermined grid-like structure

infinitesimally short distance between points

but: some form of discretization still required



Overview: what's the plan?

- Discrete:
  - PyTorch Geometric Temporal (Rozemberczki et al., 2021)
  - Example architecture from PyG Temporal (EvolveGCN – Pareja et al., 2020)
- Continuous:
  - TGNN for Deep Learning on Dynamic Graphs (Rossi et al., 2020)

Contents

1. Overview
2. **Discrete Time Approaches**
3. Continuous Time Approaches

PyTorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models

Benedek Rozemberczki*
AstraZeneca
United Kingdom
benedek.rozemberczki@astrazeneca.com

Paul Scherer
University of Cambridge
United Kingdom
pms69@cam.ac.uk

Yixuan He
University of Oxford
United Kingdom
yixuan.he@stats.ox.ac.uk

George Panagopoulos
École Polytechnique
France
george.panagopoulos@polytechnique.edu

Alexander Riedel
Ernst-Abbe University for Applied Sciences
Germany
alexander.riedel@eah-jena.de

Maria Astefanoaei
IT University of Copenhagen
Denmark
msia@itu.dk

Oliver Kiss
Central European University
Hungary
kiss_oliver@phd.ceu.edu

Ferenc Beres
ELKH SZTAKI
Hungary
beres@sztaki.hu

Guzmán López
Tryolabs
Uruguay
guzman@tryolabs.com

Nicolas Collignon
Pedal Me
United Kingdom
nicolas@pedalme.co.uk

Rik Sarkar
The University of Edinburgh
United Kingdom
rsarkar@inf.ed.ac.uk

Model	Temporal Layer	GNN Layer	Proximity Order	Multi Type
DCRNN [32]	GRU	DiffConv	Higher	False
GConvGRU [51]	GRU	Chebyshev	Lower	False
GConvLSTM [51]	LSTM	Chebyshev	Lower	False
GC-LSTM [10]	LSTM	Chebyshev	Lower	True
DyGrAE [54, 55]	LSTM	GGCN	Higher	False
LRGCN [31]	LSTM	RGCN	Lower	False
EGCN-H [39]	GRU	GCN	Lower	False
EGCN-O [39]	LSTM	GCN	Lower	False
T-GCN [65]	GRU	GCN	Lower	False
A3T-GCN [68]	GRU	GCN	Lower	False
AGCRN [4]	GRU	Chebyshev	Higher	False
MPNN LSTM [38]	LSTM	GCN	Lower	False
STGCN [63]	Attention	Chebyshev	Higher	False
ASTGCN [22]	Attention	Chebyshev	Higher	False
MSTGCN [22]	Attention	Chebyshev	Higher	False
GMAN [66]	Attention	Custom	Lower	False
MTGNN [61]	Attention	Custom	Higher	False
AAGCN [52]	Attention	Custom	Higher	False

Rozemberczki et al. (2021)

EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs

Aldo Pareja[1,2*], **Giacomo Domeniconi**[1,2*], **Jie Chen**[1,2†], **Tengfei Ma**[1,2], **Toyotaro Suzumura**[1,2], **Hiroki Kanezashi**[1,2], **Tim Kaler**[1,3] and **Charles E. Leisersen**[1,3]

[1]MIT-IBM Watson AI Lab, [2]IBM Research, [3]MIT CSAIL

{Aldo.Pareja, Giacomo.Domeniconi1}@ibm.com, chenjie@us.ibm.com
Tengfei.Ma1@ibm.com, {tsuzumura, hirokik}@us.ibm.com, {tfk, cel}@mit.edu

Abstract

Graph representation learning resurges as a trending research subject owing to the widespread use of deep learning for Euclidean data, which inspire various creative designs of neural networks in the non-Euclidean domain, particularly graphs. With the success of these graph neural networks (GNN) in the static setting, we approach further practical scenarios where the graph dynamically evolves. For this case, combining the GNN with a recurrent neural network (RNN, broadly speaking) is a natural idea. Existing approaches typically learn one single graph model for all the graphs, by using the RNN to capture the dynamism of the output node embeddings and to implicitly regulate the graph model. In this work, we propose a different approach, coined EvolveGCN, that uses the RNN to evolve the graph

Grover and Leskovec, 2016] on both the node and the graph level, now parameterized by deep neural networks [Bruna *et al.*, 2014; Duvenaud *et al.*, 2015; Defferrard *et al.*, 2016; Li *et al.*, 2016; Gilmer *et al.*, 2017; Kipf and Welling, 2017; Hamilton *et al.*, 2017; Jin *et al.*, 2017; Chen *et al.*, 2018; Veličković *et al.*, 2018].

These neural network models generally focus on a given, static graph. In real-life applications, however, often one encounters a dynamically evolving graph. For example, users of a social network develop friendship over time; hence, the vectorial representation of the users should be updated accordingly to reflect the temporal evolution of their social relationship. Similarly, a citation network of scientific articles is constantly enriched due to frequent publications of new work citing prior art. Thus, the influence, and even sometimes the categorization, of an article varies along time. Update of the node embeddings to reflect this variation is desired. In financial networks, transactions naturally come with time stamps

Discrete time: EvolveGCN

1: **function** $[H_t^{(l+1)}, W_t^{(l)}] = \mathrm{EGCU}(A_t, H_t^{(l)}, W_{t-1}^{(l)})$
2: $\quad W_t^{(l)} = \mathrm{GRU}(H_t^{(l)}, W_{t-1}^{(l)})$
3: $\quad H_t^{(l+1)} = \mathrm{GCONV}(A_t, H_t^{(l)}, W_t^{(l)})$
4: **end function**

$H_t^{(0)} = X_t$

$H_t^{(l)}$: node embedding matrix at time t and level l

$W_t^{(l)}$: weight matrix at time t and level l

A_t: adjacency matrix at time t

Discrete time: EvolveGCN - GCONV

1: **function** $[H_t^{(l+1)}, W_t^{(l)}] = \mathrm{EGCU}(A_t, H_t^{(l)}, W_{t-1}^{(l)})$
2: $\quad W_t^{(l)} = \mathrm{GRU}(H_t^{(l)}, W_{t-1}^{(l)})$
3: $\quad H_t^{(l+1)} = \mathrm{GCONV}(A_t, H_t^{(l)}, W_t^{(l)})$
4: **end function**

$H_t^{(l)}$: node embedding matrix at time t and level l

$W_t^{(l)}$: weight matrix at time t and level l

A_t: adjacency matrix at time t

Discrete time: EvolveGCN - GCONV

$$H_t^{(l+1)} = \mathrm{GCONV}(A_t, H_t^{(l)}, W_t^{(l)}) := \sigma(\widehat{A}_t H_t^{(l)} W_t^{(l)}), \quad (1)$$

where $\widehat{A}_t$ is a normalization of A_t defined as (omitting time index for clarity):

$$\widehat{A} = \widetilde{D}^{-\frac{1}{2}} \widetilde{A} \widetilde{D}^{-\frac{1}{2}}, \quad \widetilde{A} = A + I, \quad \widetilde{D} = \mathrm{diag}\left(\sum_j \widetilde{A}_{ij} \right),$$

Kipf & Welling (2017)

Non-existing node ≡ dangling node with zero degree.
It does not affect information flow in graph convolutions

Discrete time: EvolveGCN - GRU

1: **function** $[H_t^{(l+1)}, W_t^{(l)}] = \mathrm{EGCU}(A_t, H_t^{(l)}, W_{t-1}^{(l)})$
2: $\quad W_t^{(l)} = \mathrm{GRU}(H_t^{(l)}, W_{t-1}^{(l)})$
3: $\quad H_t^{(l+1)} = \mathrm{GCONV}(A_t, H_t^{(l)}, W_t^{(l)})$
4: **end function**

$H_t^{(l)}$: node embedding matrix at time t and level l

$W_t^{(l)}$: weight matrix at time t and level l

A_t: adjacency matrix at time t

Discrete time: EvolveGCN - GRU

$$W_t^{(l)} = \mathrm{GRU}(H_t^{(l)}, W_{t-1}^{(l)})$$
$$:= g(summarize(H_t^{(l)}, \#col(W_{t-1}^{(l)}))^T, W_{t-1}^{(l)}),$$

function $H_t = g(X_t, H_{t-1})$

$Z_t = \mathrm{sigmoid}(W_Z X_t + U_Z H_{t-1} + B_Z)$

$R_t = \mathrm{sigmoid}(W_R X_t + U_R H_{t-1} + B_R)$

$\widetilde{H}_t = \tanh(W_H X_t + U_H (R_t \circ H_{t-1}) + B_H)$

$H_t = (1 - Z_t) \circ H_{t-1} + Z_t \circ \widetilde{H}_t$

end function

H_t: hidden state at point t

Z_t: output gate

R_t: reset gate

$\tilde{H}_t$: pre-output

$\circ$: element-wise multiplication

Discrete time: EvolveGCN - GRU

$$W_t^{(l)} = \mathrm{GRU}(H_t^{(l)}, W_{t-1}^{(l)})$$
$$:= g(summarize(H_t^{(l)}, \#col(W_{t-1}^{(l)}))^T, W_{t-1}^{(l)}),$$

- **Goal**: treat node embeddings $H_t^{(l)}$ as GRU input and weight matrix $W_t^{(l)}$ as new GRU hidden state
- **Problem**: #col($H_t^{(l)}$) ≠ #col($W_t^{(l)}$) → $H_t^{(l)} \in \mathbb{R}^{n \times h_l}, W_t^{(l)} \in \mathbb{R}^{h_l \times d_l}$
- **Solution**: summarize embedding matrix $H_t^{(l)}$ into one with *k* columns, where k = #col($W_t^{(l)}$)

function $Z_t = summarize(X_t, k)$
  $y_t = X_t p / \|p\|$
  $i_t = \text{top-indices}(y_t, k)$
  $Z_t = [X_t \circ \tanh(y_t)]_{i_t}$
end function

Cangea et al. (2018)

- p is a parameter vector, independent of index *t*, but which may vary for different GCONV layers
- y_t is a vector of weights for each row
- i_t takes the top *k* indices of the weight vector

Rozemberczki et al. (2021)

Discrete time: EvolveGCN summary

1: **function** $[H_t^{(l+1)}, W_t^{(l)}] = \mathrm{EGCU}(A_t, H_t^{(l)}, W_{t-1}^{(l)})$
2: $\quad W_t^{(l)} = \mathrm{GRU}(H_t^{(l)}, W_{t-1}^{(l)})$
3: $\quad H_t^{(l+1)} = \mathrm{GCONV}(A_t, H_t^{(l)}, W_t^{(l)})$
4: **end function**

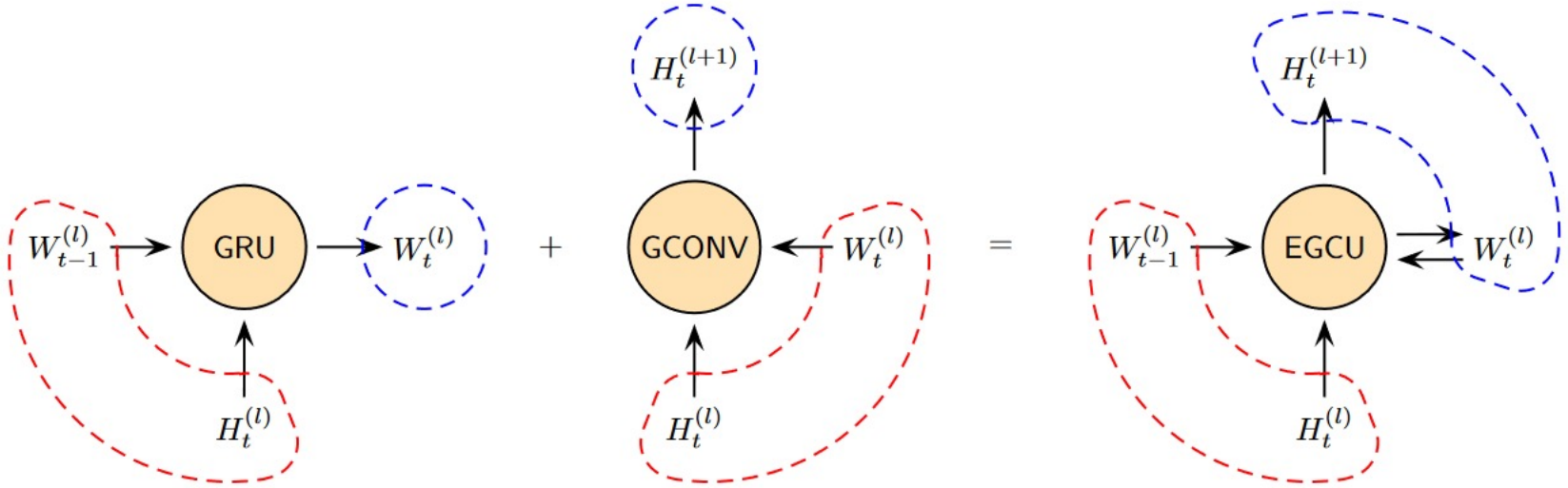


Figure 1: Left: gated recurrent unit; middle: graph convolution unit; right: evolving graph convolution unit. Red region denotes input and blue region denotes output. Time t progresses from left to right, whereas neural network layers l are built up from bottom to top.

Pareja et al. (2020)

Discrete time: incremental / cumulative

Incremental

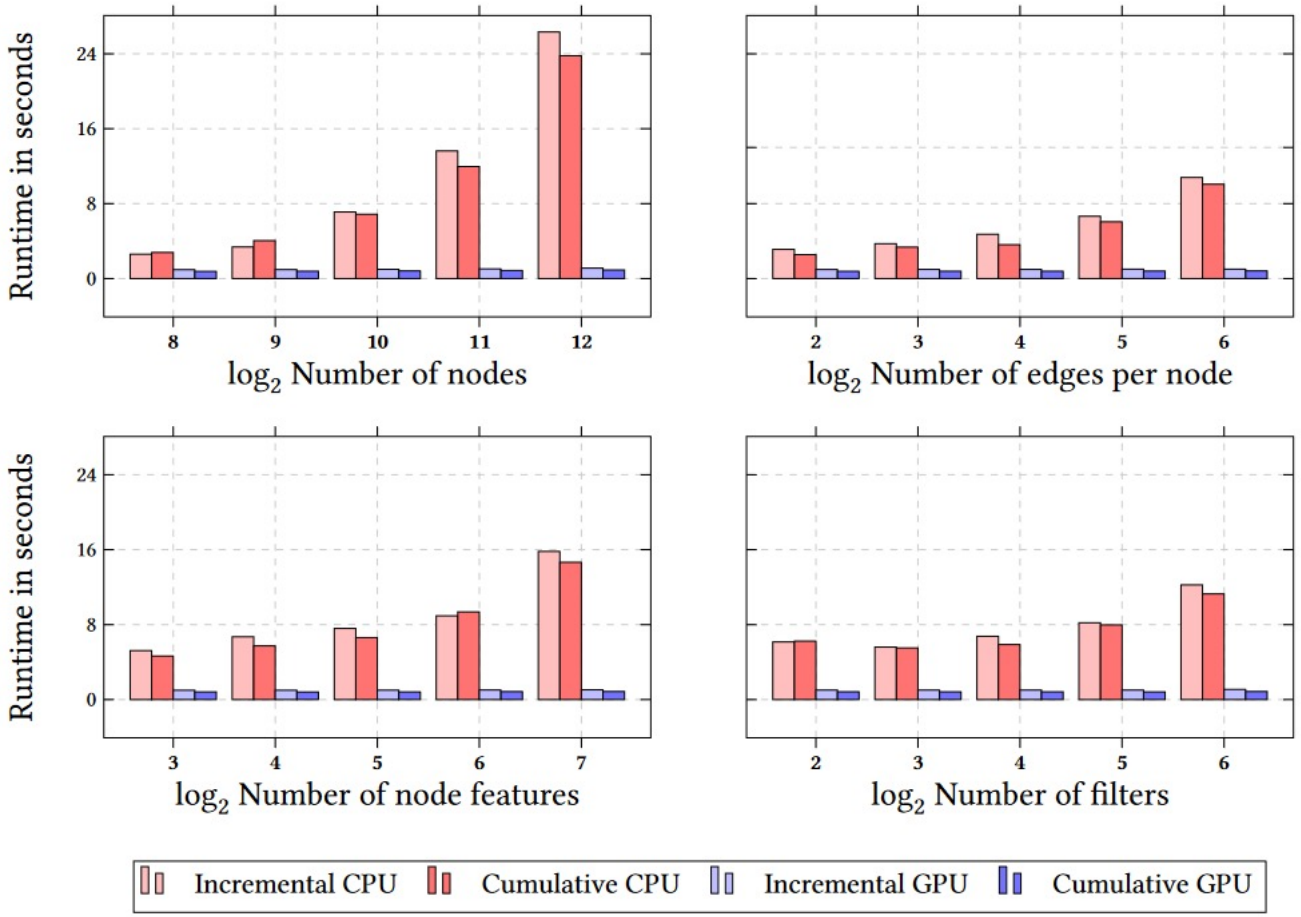
```
model = RecurrentGCN(node_features=8, filters=4)
device = torch.device('cuda')
model = model.to(device)

optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
model.train()

for epoch in range(200):
    for snapshot in train:
        snapshot = snapshot.to(device)
        y_hat = model(snapshot.x,
                      snapshot.edge_index,
                      snapshot.edge_attr)
        cost = torch.mean((y_hat-snapshot.y)**2)
        cost.backward()
        optimizer.step()
        optimizer.zero_grad()
```

Cumulative

```
model = RecurrentGCN(node_features=8, filters=4)

optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

model.train()

for epoch in range(200):
    cost = 0
    for time, snapshot in enumerate(train):
        y_hat = model(snapshot.x,
                      snapshot.edge_index,
                      snapshot.edge_attr)
        cost = cost + torch.mean((y_hat-snapshot.y)**2)
    cost = cost / (time+1)
    cost.backward()
    optimizer.step()
    optimizer.zero_grad()
```

Rozemberczki et al. (2021)

Discrete time: cumulative / incremental



Rozemberczki et al. (2021)

Discrete time: performance (MSE)

Table 5: The predictive performance of spatiotemporal neural networks evaluated by average mean squared error. We report average performances calculated from 10 experimental repetitions with standard deviations around the average mean squared error calculated on 10% forecasting horizons. We use the incremental and cumulative backpropagation strategies.

	Chickenpox Hungary		Twitter Tennis RG		PedalMe London		Wikipedia Math	
	Incremental	Cumulative	Incremental	Cumulative	Incremental	Cumulative	Incremental	Cumulative
DCRNN [32]	1.124 ± 0.015	1.123 ± 0.014	2.049 ± 0.023	2.043 ± 0.016	1.463 ± 0.019	1.450 ± 0.024	**0.679 ± 0.020**	**0.803 ± 0.018**
GConvGRU [51]	1.128 ± 0.011	1.132 ± 0.023	2.051 ± 0.020	2.007 ± 0.022	1.622 ± 0.032	1.944 ± 0.013	0.657 ± 0.015	0.837 ± 0.021
GConvLSTM [51]	1.121 ± 0.014	1.119 ± 0.022	2.049 ± 0.024	2.007 ± 0.012	1.442 ± 0.028	1.433 ± 0.020	0.777 ± 0.021	0.868 ± 0.018
GC-LSTM [10]	1.115 ± 0.014	1.116 ± 0.023	2.053 ± 0.024	2.032 ± 0.015	**1.455 ± 0.023**	1.468 ± 0.025	0.779 ± 0.023	0.852 ± 0.016
DyGrAE [54, 55]	1.120 ± 0.021	1.118 ± 0.015	**2.031 ± 0.006**	2.007 ± 0.004	**1.455 ± 0.031**	1.456 ± 0.019	0.773 ± 0.009	0.816 ± 0.016
EGCN-H [39]	**1.113 ± 0.016**	**1.104 ± 0.024**	2.040 ± 0.018	**2.006 ± 0.008**	1.467 ± 0.026	1.436 ± 0.017	0.775 ± 0.022	0.857 ± 0.022
EGCN-O [39]	1.124 ± 0.009	1.119 ± 0.020	2.055 ± 0.020	2.010 ± 0.014	1.491 ± 0.024	**1.430 ± 0.023**	0.750 ± 0.014	0.823 ± 0.014
A3T-GCN[68]	1.114 ± 0.008	1.119 ± 0.018	2.045 ± 0.021	2.008 ± 0.016	1.469 ± 0.027	1.475 ± 0.029	0.781 ± 0.011	0.872 ± 0.017
T-GCN [65]	1.117 ± 0.011	1.111 ± 0.022	2.045 ± 0.027	2.008 ± 0.017	1.479 ± 0.012	1.481 ± 0.029	0.764 ± 0.011	0.846 ± 0.020
MPNN LSTM [38]	1.116 ± 0.023	1.129 ± 0.021	2.053 ± 0.041	2.007 ± 0.010	1.485 ± 0.028	1.458 ± 0.013	0.795 ± 0.010	0.905 ± 0.017
AGCRN [4]	1.120 ± 0.010	1.116 ± 0.017	2.039 ± 0.022	2.010 ± 0.009	1.469 ± 0.030	1.465 ± 0.026	0.788 ± 0.011	0.832 ± 0.020

TL;DR – there is no silver bullet!

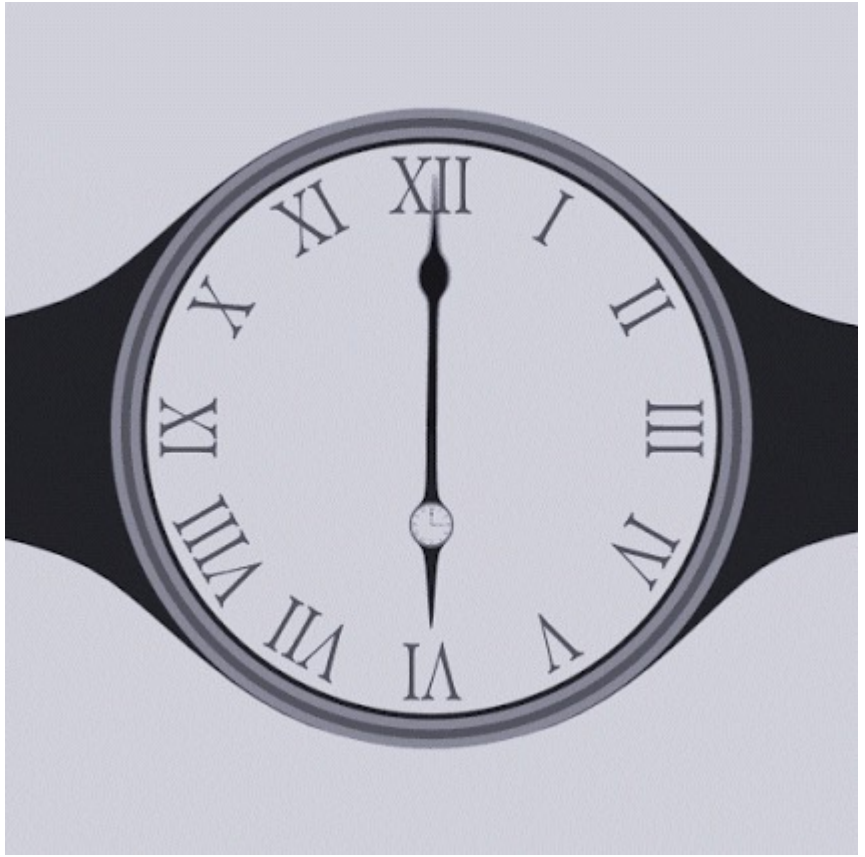
Rozemberczki et al. (2021)

Contents

1. Overview

2. Discrete Time Approaches

3. **Continuous Time Approaches**



TEMPORAL GRAPH NETWORKS FOR DEEP LEARNING ON DYNAMIC GRAPHS

Emanuele Rossi* Twitter

Ben Chamberlain Twitter

Fabrizio Frasca Twitter

Davide Eynard Twitter

Federico Monti Twitter

Michael Bronstein Twitter

ABSTRACT

Graph Neural Networks (GNNs) have recently become increasingly popular due to their ability to learn complex systems of relations or interactions. These arise in a broad spectrum of problems ranging from biology and particle physics to social networks and recommendation systems. Despite the plethora of different models for deep learning on graphs, few approaches have been proposed for dealing with graphs that are dynamic in nature (e.g. evolving features or connectivity over time). We present Temporal Graph Networks (TGNs), a generic, efficient framework for deep learning on dynamic graphs represented as sequences of timed events. Thanks to a novel combination of memory modules and graph-based operators, TGNs significantly outperform previous approaches while being more computationally efficient. We furthermore show that several previous models for learning on dynamic graphs can be cast as specific instances of our framework. We perform a detailed ablation study of different components of our framework and devise the best configuration that achieves state-of-the-art performance on several transductive and inductive prediction tasks for dynamic graphs.

Continuous time: TGNN motivation

- Twitter:
  - Users tweet at highly irregular rates
  - Each tweet has a timestamp
  - Users create and delete accounts

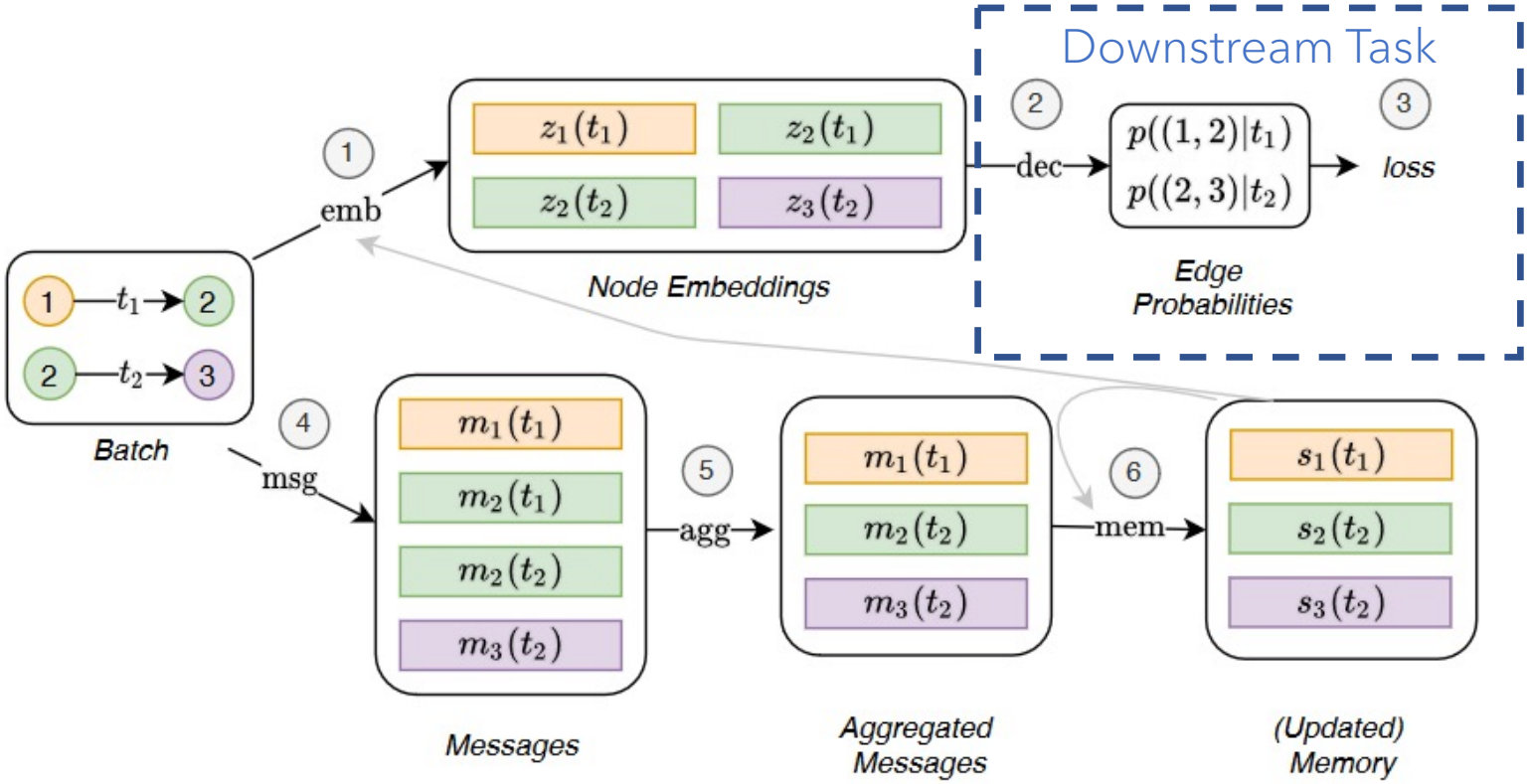
Continuous time: TGNN notation

- Temporal Graph: sequence of time-stamped events

$$\mathcal{G} = \{x(t_1), x(t_2), \ldots\} \quad \text{for} \quad 0 \leq t_1 \leq t_2 \leq \ldots$$

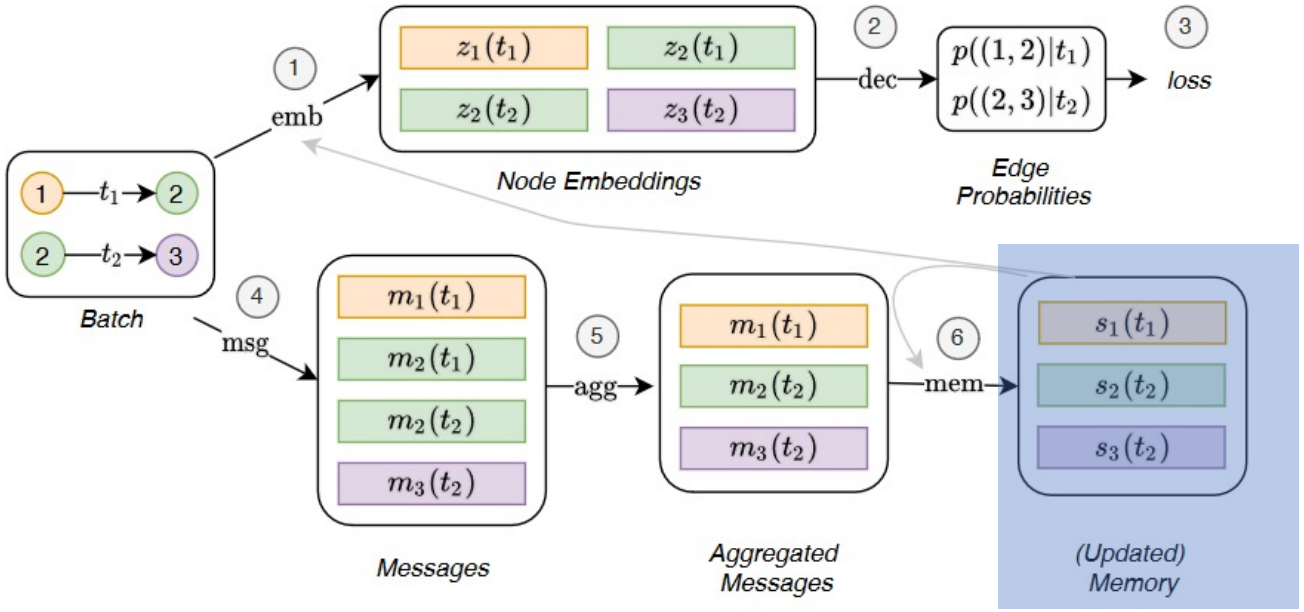
- Two types of event $\boldsymbol{x(t)}$:
  - **Node**-wise event: $\boldsymbol{v_i(t)}$, where *i* is the node index, $\boldsymbol{v_i}$ the vector attribute
  - **Interaction** event between nodes *i* and *j* denoted by temporal edge $\boldsymbol{e_{ij}(t)}$
- Temporal set of nodes: $\mathcal{V}(T) = \{i \ : \ \exists \mathbf{v}_i(t) \in \mathcal{G}, t \in T\}$
- Temporal set of edges: $\mathcal{E}(T) = \{(i,j) \ : \ \exists \mathbf{e}_{ij}(t) \in \mathcal{G}, t \in T\}$
- Temporal neighbors of node *i*: $\mathcal{N}_i(T) = \{j \ : \ (i,j) \in \mathcal{E}(T)\}$

Continuous time: TGNN architecture



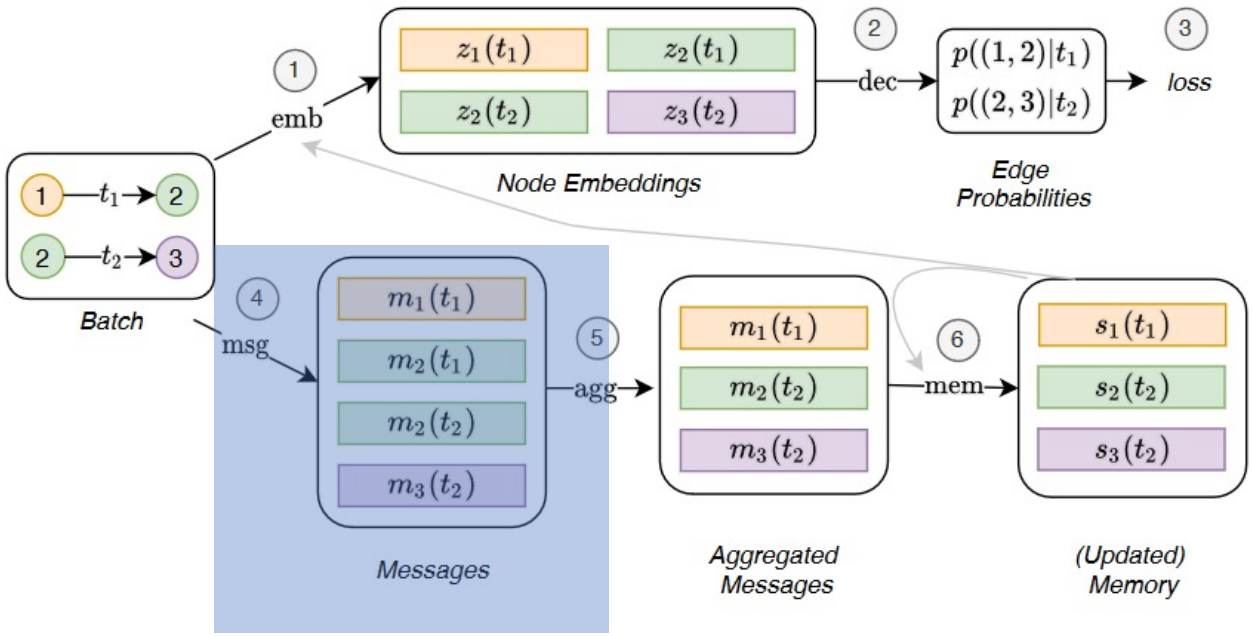
Rossi et al. (2020)

Continuous time: TGNN - Memory



- The memory at time *t* consists of a vector of states $s_i(t)$ for each node *i* seen so far
- Represents history in **compressed** format
- Helps memorize **long-term** dependencies
- New node initialized to **zero** vector
- The memory is **not trainable**! It changes even after finishing training the model

Continuous time: TGNN - Messages



- Interaction event for nodes *i* and *j*

$$\mathbf{m}_i(t) = \mathrm{msg}_\mathrm{s}\left(\mathbf{s}_i(t^-), \mathbf{s}_j(t^-), \Delta t, \mathbf{e}_{ij}(t)\right),$$

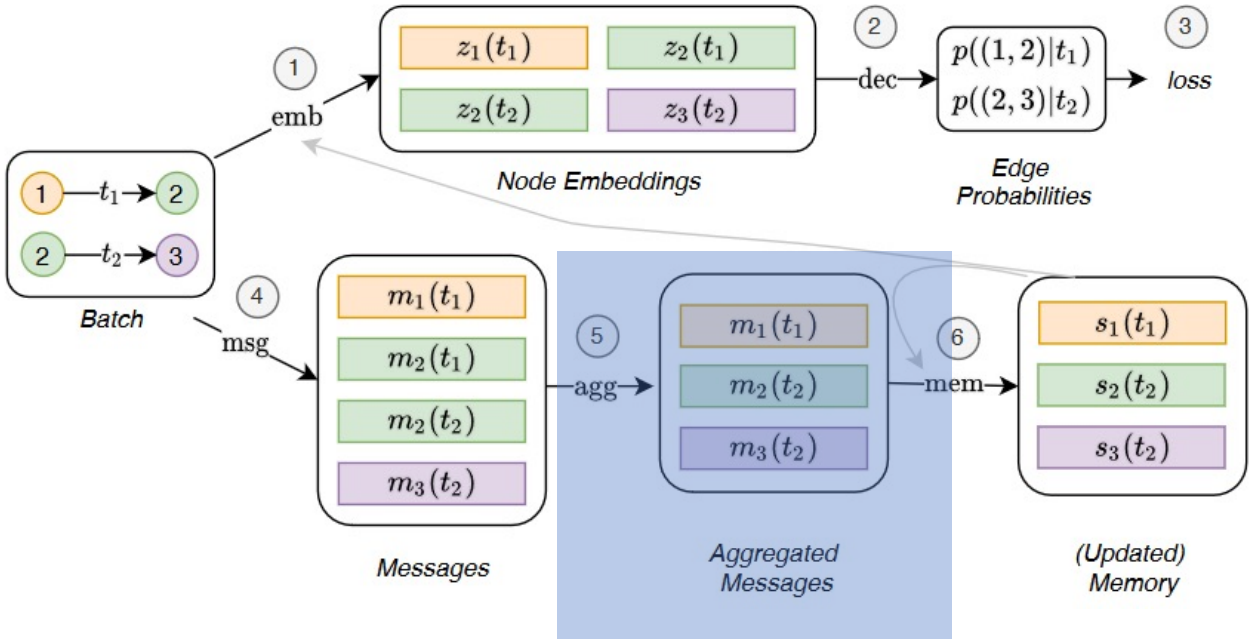
$$\mathbf{m}_j(t) = \mathrm{msg}_\mathrm{d}\left(\mathbf{s}_j(t^-), \mathbf{s}_i(t^-), \Delta t, \mathbf{e}_{ij}(t)\right)$$

- Node-wise event for node *i*

$$\mathbf{m}_i(t) = \mathrm{msg}_\mathrm{n}\left(\mathbf{s}_i(t^-), t, \mathbf{v}_i(t)\right)$$

- $s_i(t^-)$ is the memory of node *i* just before time *t*
- msg_s, msg_d, msg_n are learnable functions
- In the paper, they just **concatenate** the inputs of the messages

Continuous time: TGNN - Aggregation

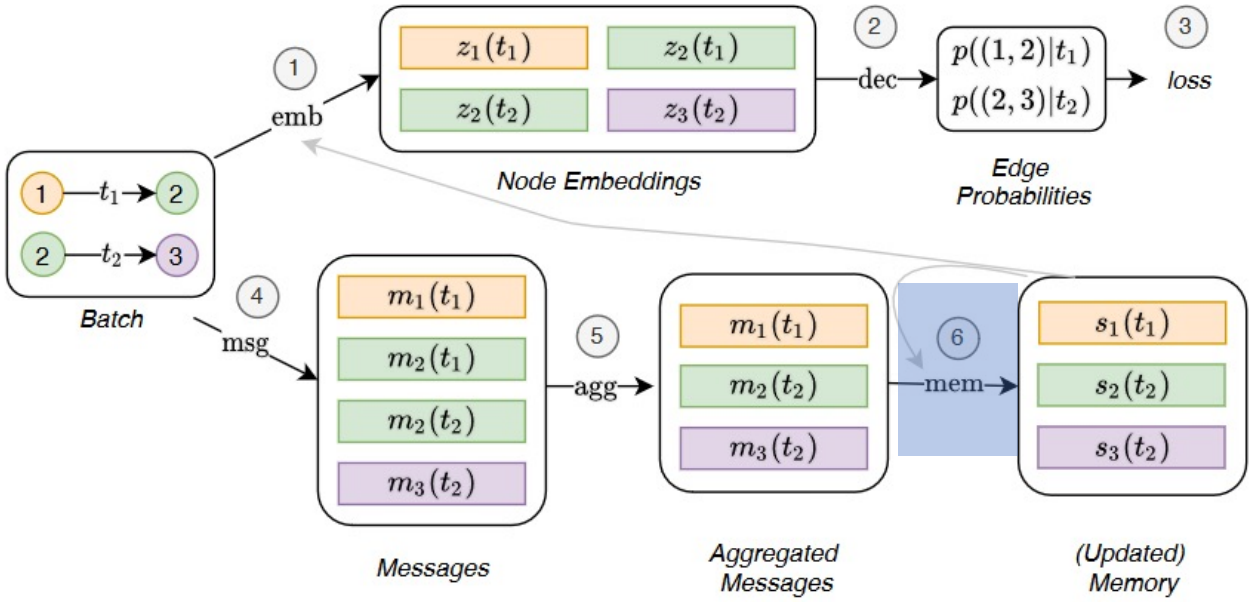


- When doing batch processing we might end up with multiple messages for node *i*
- We want a **single message** for each node
- Define an aggregation function:

$$\bar{\mathbf{m}}_i(t) = \text{agg}\left(\mathbf{m}_i(t_1), \ldots, \mathbf{m}_i(t_b)\right)$$

- In the paper, they use non-learnable *agg*:
  - Most recent message
  - Mean message

Continuous time: TGNN - Updating

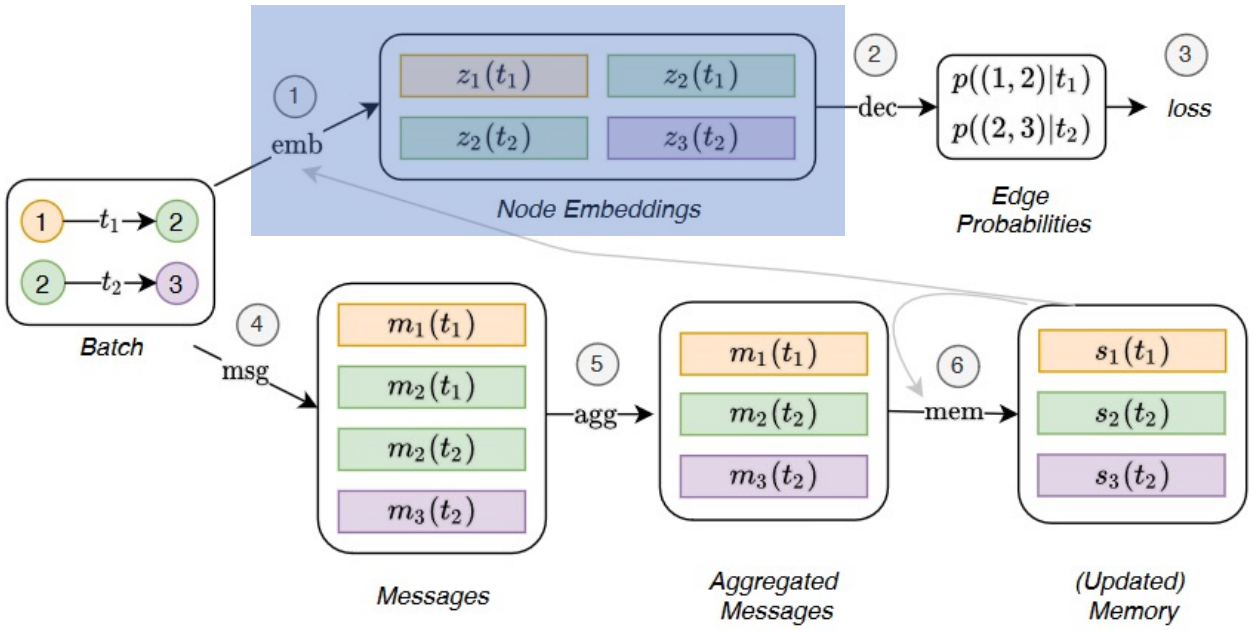


- Update memory given aggregated message and old memory state:

$$\mathbf{s}_i(t) = \text{mem}\left(\bar{\mathbf{m}}_i(t), \mathbf{s}_i(t^-)\right)$$

- *mem* is a learnable update function, e.g., GRU or LSTM
- For **interaction** events, both nodes are updated
- For **node events** only the relevant node is updated

Continuous time: TGNN - Embedding



- Generates temporal embedding $z_i(t)$ of node *i* at time *t*.
- Avoids *memory staleness* when node *i* is not involved in any event
- Uses learnable function *h*:

$$\mathbf{z}_i(t) = \mathrm{emb}(i,t) = \sum_{j \in n_i^k([0,t])} h\left(\mathbf{s}_i(t), \mathbf{s}_j(t), \mathbf{e}_{ij}, \mathbf{v}_i(t), \mathbf{v}_j(t)\right)$$

- Examples for *h*:
  - Identity: $\mathrm{emb}(i,t) = \mathbf{s}_i(t)$
  - Time proj.: $\mathrm{emb}(i,t) = (1 + \Delta t\, \mathbf{w}) \circ \mathbf{s}_i(t)$

Continuous time: TGNN - Embedding

Temporal Graph Attention (attn): A series of L graph attention layers compute i's embedding by aggregating information from its L-hop temporal neighborhood.

The input to the l-th layer is i's representation $\mathbf{h}_i^{(l-1)}(t)$, the current timestamp t, i's neighborhood representation $\{\mathbf{h}_1^{(l-1)}(t), \ldots, \mathbf{h}_N^{(l-1)}(t)\}$ together with timestamps $t_1, \ldots, t_N$ and features $\mathbf{e}_{i1}(t_1), \ldots, \mathbf{e}_{iN}(t_N)$ for each of the considered interactions which form an edge in i's temporal neighborhood:

$$\mathbf{h}_i^{(l)}(t) = \text{MLP}^{(l)}(\mathbf{h}_i^{(l-1)}(t) \,\|\, \tilde{\mathbf{h}}_i^{(l)}(t)), \qquad \mathbf{h}_j^{(0)}(t) = \mathbf{s}_j(t) + \mathbf{v}_j(t) \tag{5}$$

$$\tilde{\mathbf{h}}_i^{(l)}(t) = \text{MultiHeadAttention}^{(l)}(\mathbf{q}^{(l)}(t), \mathbf{K}^{(l)}(t), \mathbf{V}^{(l)}(t)), \tag{6}$$

$$\mathbf{q}^{(l)}(t) = \mathbf{h}_i^{(l-1)}(t) \,\|\, \phi(0), \tag{7}$$

$$\mathbf{K}^{(l)}(t) = \mathbf{V}^{(l)}(t) = \mathbf{C}^{(l)}(t), \tag{8}$$

$$\mathbf{C}^{(l)}(t) = [\mathbf{h}_1^{(l-1)}(t) \,\|\, \mathbf{e}_{i1}(t_1) \,\|\, \phi(t - t_1), \ldots, \mathbf{h}_N^{(l-1)}(t) \,\|\, \mathbf{e}_{iN}(t_N) \,\|\, \phi(t - t_N)]. \tag{9}$$

$$\mathbf{z}_i(t) = \text{emb}(i, t) = \mathbf{h}_i^{(L)}(t)$$

Continuous time: TGNN - Embedding

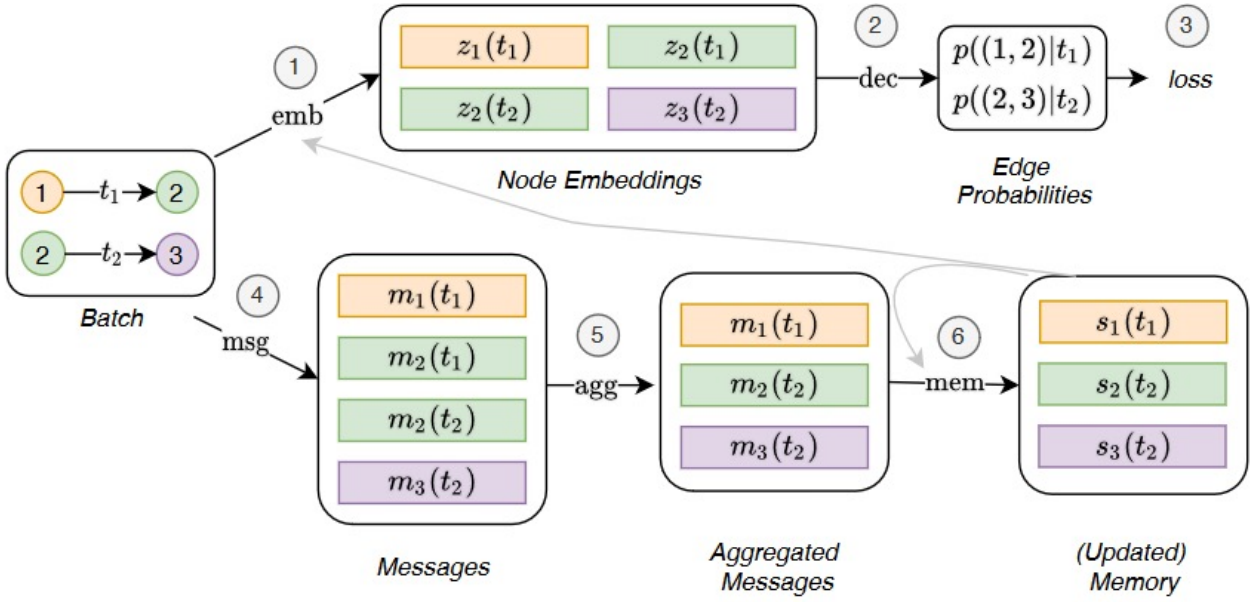
Temporal Graph Sum (sum): A simpler and faster aggregation over the graph:

$$\mathbf{h}_i^{(l)}(t) = \mathbf{W}_2^{(l)}(\mathbf{h}_i^{(l-1)}(t) \,\|\, \tilde{\mathbf{h}}_i^{(l)}(t)),$$
$$\tilde{\mathbf{h}}_i^{(l)}(t) = \mathrm{ReLu}\Big(\sum_{j \in n_i([0,t])} \mathbf{W}_1^{(l)}(\mathbf{h}_j^{(l-1)}(t) \,\|\, \mathbf{e}_{ij} \,\|\, \phi(t - t_j))\Big).$$

$$\mathbf{z}_i(t) = \mathrm{emb}(i,t) = \mathbf{h}_i^{(L)}(t)$$

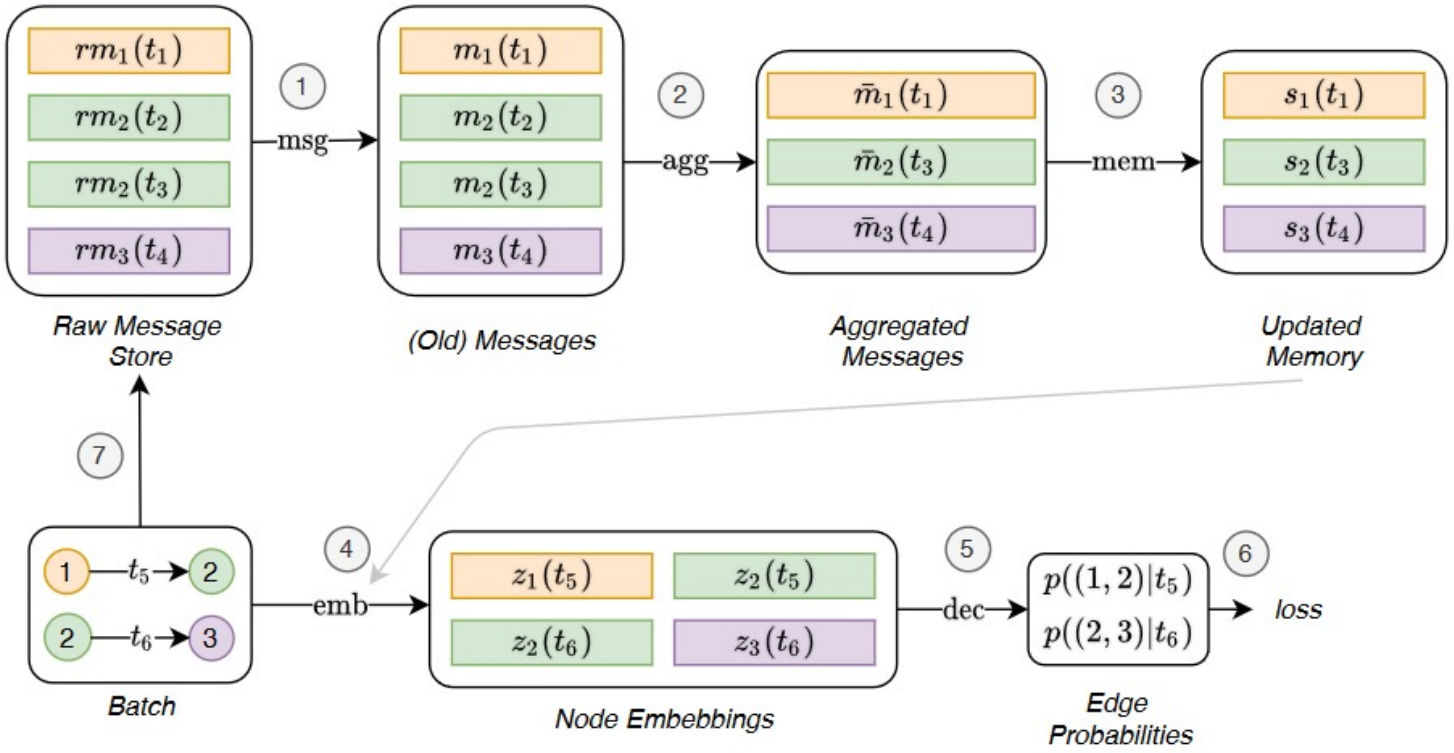
Continuous time: TGNN - Training

- Problem:
  - Memory-related functions (msg, agg, mem) do not influence the loss directly and have no gradient
  - Therefore, the memory must be updated *before* predicting batch interactions
  - This causes *information leakage* – storing interactions before predicting the same interactions



Continuous time: TGNN - Training

- Solution:
  - Update the memory with from messages previous batches stored in *raw message store*
  - After an interaction concerning node *i* the memory is updated with the message from the previous batch. The message from the present batch in stored in the *raw message store*



Continuous time: TGNN - Comparison

Table 1: Previous models for deep learning on continuous-time dynamic graphs are specific case of our TGN framework. Shown are multiple variants of TGN used in our ablation studies. *method* (l,n) refers to graph convolution using l layers and n neighbors. †uses t-batches. * uses uniform sampling of neighbors, while the default is sampling the most recent neighbors. ‡message aggregation not explained in the paper. ‖ uses a summary of the destination node neighborhood (obtained through graph attention) as additional input to the message function.

	Mem.	Mem. Updater	Embedding	Mess. Agg.	Mess. Func.
Jodie	node	RNN	time	—†	id
TGAT	—	—	attn (2l, 20n)*	—	—
DyRep	node	RNN	id	—‡	attn‖
TGN-attn	node	GRU	attn (1l, 10n)	last	id
TGN-2l	node	GRU	attn (2l, 10n)	last	id
TGN-no-mem	—	—	attn (1l, 10n)	—	—
TGN-time	node	GRU	time	last	id
TGN-id	node	GRU	id	last	id
TGN-sum	node	GRU	sum (1l, 10n)	last	id
TGN-mean	node	GRU	attn (1l, 10n)	mean	id

Rossi et al. (2020)

Continuous time: TGNN - Performance

Table 2: Average Precision (%) for future edge prediction task in transductive and inductive settings. **First**, **Second**, **Third** best performing method. *Static graph method. †Does not support inductive.

	Wikipedia		Reddit		Twitter	
	Transductive	Inductive	Transductive	Inductive	Transductive	Inductive
GAE*	91.44 ± 0.1	†	93.23 ± 0.3	†	—	†
VAGE*	91.34 ± 0.3	†	92.92 ± 0.2	†	—	†
DeepWalk*	90.71 ± 0.6	†	83.10 ± 0.5	†	—	†
Node2Vec*	91.48 ± 0.3	†	84.58 ± 0.5	†	—	†
GAT*	**94.73** ± 0.2	91.27 ± 0.4	97.33 ± 0.2	95.37 ± 0.3	67.57 ± 0.4	62.32 ± 0.5
GraphSAGE*	93.56 ± 0.3	91.09 ± 0.3	97.65 ± 0.2	**96.27** ± 0.2	65.79 ± 0.6	60.13 ± 0.6
CTDNE	92.17 ± 0.5	†	91.41 ± 0.3	†	—	†
Jodie	94.62 ± 0.5	**93.11** ± 0.4	97.11 ± 0.3	94.36 ± 1.1	**85.20** ± 2.4	**79.83** ± 2.5
TGAT	**95.34** ± 0.1	**93.99** ± 0.3	**98.12** ± 0.2	**96.62** ± 0.3	70.02 ± 0.6	66.35 ± 0.8
DyRep	94.59 ± 0.2	92.05 ± 0.3	**97.98** ± 0.1	95.68 ± 0.2	**83.52** ± 3.0	**78.38** ± 4.0
TGN-attn	**98.46** ± 0.1	**97.81** ± 0.1	**98.70** ± 0.1	**97.55** ± 0.1	**94.52** ± 0.5	**91.37** ± 1.1

Rossi et al. (2020)

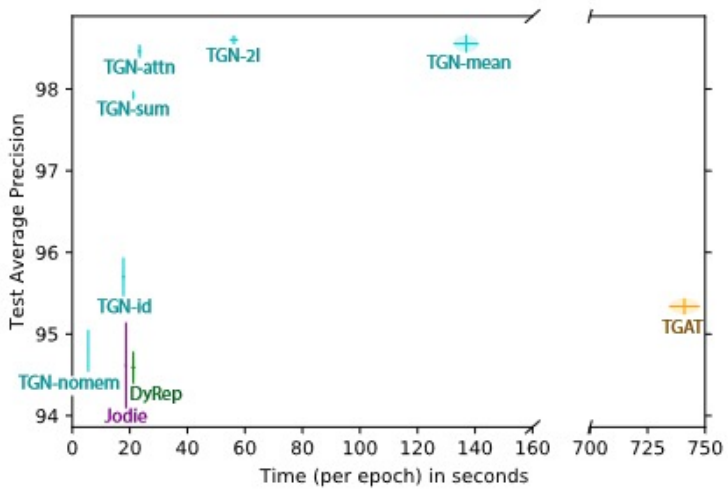
Continuous time: TGNN - Performance

Table 3: ROC AUC % for the dynamic node classification. *Static graph method.

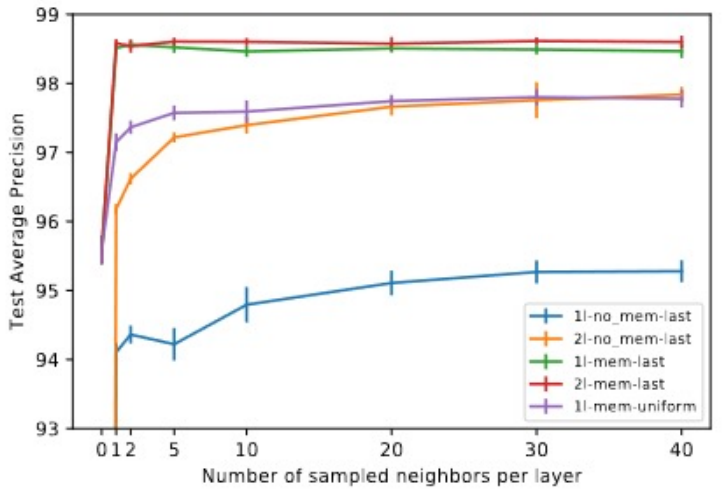
	Wikipedia	Reddit
GAE*	74.85 ± 0.6	58.39 ± 0.5
VAGE*	73.67 ± 0.8	57.98 ± 0.6
GAT*	82.34 ± 0.8	**64.52** ± 0.5
GraphSAGE*	82.42 ± 0.7	61.24 ± 0.6
CTDNE	75.89 ± 0.5	59.43 ± 0.6
JODIE	**84.84** ± 1.2	61.83 ± 2.7
TGAT	83.69 ± 0.7	**65.56** ± 0.7
DyRep	**84.59** ± 2.2	62.91 ± 2.4
TGN-attn	**87.81** ± 0.3	**67.06** ± 0.9

Rossi et al. (2020)

Continuous time: TGNN - Performance



(a) Tradeoff between accuracy (test average precision in %) and speed (time per epoch in sec) of different models.



(b) Performance of methods with different number of layers and with or without memory when sampling an increasing number of neighbors. *Last* and *uniform* refer to neighbor sampling strategy.

Figure 3: Ablation studies on the Wikipedia dataset for the transductive setting of the future edge prediction task. Means and standard deviations were computed over 10 runs.

Rossi et al. (2020)

Summary

- Make sure to compare temporal GNNs with more classical approaches - e.g. autoregressive models
- Temporal neural networks are particularly useful beyond just node prediction
- Models can get more powerful, but hard problems don’t get easier!

Questions?

flaviomorelli.com

@mexiamorelli

linkedin.com/in/mexiamorelli

Backup

Table 4: Properties and granularity of the spatiotemporal datasets introduced in the paper with information about the number of time periods (T) and spatial units ($|V|$).

Dataset	Signal	Graph	Frequency	T	$\|V\|$
Chickenpox Hungary	Temporal	Static	Weekly	522	20
Windmill Large	Temporal	Static	Hourly	17,472	319
Windmill Medium	Temporal	Static	Hourly	17,472	26
Windmill Small	Temporal	Static	Hourly	17,472	11
Pedal Me Deliveries	Temporal	Static	Weekly	36	15
Wikipedia Math	Temporal	Static	Daily	731	1,068
Twitter Tennis RG	Static	Dynamic	Hourly	120	1000
Twitter Tennis UO	Static	Dynamic	Hourly	112	1000
Covid19 England	Temporal	Dynamic	Daily	61	129
Montevideo Buses	Temporal	Static	Hourly	744	675
MTM-1 Hand Motions	Temporal	Static	1/24 Seconds	14,469	21

Rozemberczki et al. (2021)

Chickenpox Hungary. A spatiotemporal dataset about the officially reported cases of chickenpox in Hungary. The nodes are counties and edges describe direct neighbourhood relationships. The dataset covers the weeks between 2005 and 2015 without missingness.

Twitter Tennis RG and UO. Twitter mention graphs of major tennis tournaments from 2017. Each snapshot contains the graph of popular player or sport news accounts and mentions between them [5, 6]. Node labels encode the number of mentions received and vertex features are structural properties.

Pedal Me Deliveries. A dataset about the number of weekly bicycle package deliveries by Pedal Me in London during 2020 and 2021. Nodes in the graph represent geographical units and edges are proximity based mutual adjacency relationships.

Wikipedia Math. Contains Wikipedia pages about popular mathematics topics and edges describe the links from one page to another. Features describe the number of daily visits between 2019 and 2021 March.

Datasets. We use three datasets in our experiments: Wikipedia, Reddit (Kumar et al., 2019), and Twitter, which are described in detail in Appendix A.3. Our experimental setup closely follows (Xu et al., 2020) and focuses on the tasks of future edge ('link') prediction and dynamic node classification. In future edge prediction, the goal is to predict the probability of an edge occurring between two nodes at a given time. Our encoder is combined with a simple MLP decoder mapping from the concatenation of two node embeddings to the probability of the edge. We study both the transductive and inductive settings. In the transductive task, we predict future links of the nodes observed during training, whereas in the inductive tasks we predict future links of nodes never observed before. For node classification, the transductive setting is used. For all tasks and datasets we perform the same 70%-15%-15% chronological split as in Xu et al. (2020). All the results were averaged over 10 runs. Hyperparameters and additional details can be found in Appendix A.4.

Rossi et al. (2020)

Reddit and Wikipedia are bipartite interaction graphs. In the Reddit dataset, users and sub-reddits are nodes, and an interaction occurs when a user writes a post to the sub-reddit. In the Wikipedia dataset, users and pages are nodes, and an interaction represents a user editing a page. In both aforementioned datasets, the interactions are represented by text features (of a post or page edit, respectively), and labels represent whether a user is banned. Both interactions and labels are time-stamped.

The Twitter dataset is a non-bipartite graph released as part of the 2020 RecSys Challenge (Belli et al., 2020). Nodes are users and interactions are retweets. The features of an interaction are a BERT-based (Wolf et al., 2019) vector representation of the text of the retweet.

Node features are not present in any of these datasets, and we therefore assign the same zero feature vector to all nodes. Moreover, While our framework is general and in section 3.1 we showed how it can process any type of event, these three datasets only contain the edge creation (interaction) event type. Creating and evaluation of datasets with a wider variety of events is left as future work.

Rossi et al. (2020)

Table 4: Statistics of the datasets used in the experiments.

	Wikipedia	Reddit	Twitter
# Nodes	9,227	11,000	8,861
# Edges	157,474	672,447	119,872
# Edge features	172	172	768
# Edge features type	LIWC	LIWC	BERT
Timespan	30 days	30 days	7 days
Chronological Split	70%-15%-15%	70%-15%-15%	70%-15%-15%
# Nodes with dynamic labels	217	366	–

Rossi et al. (2020)