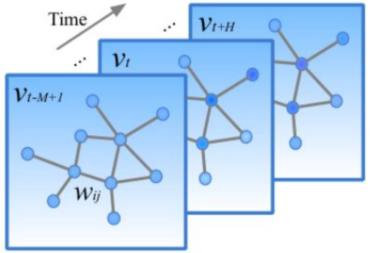
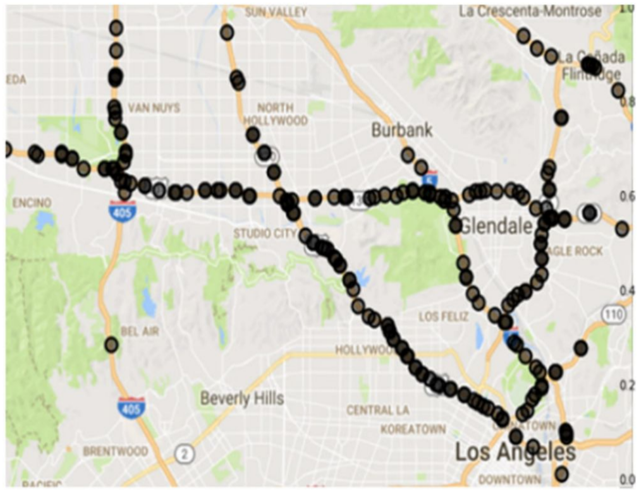
Self-Supervision on Dynamic Graphs

Group 4
Yusong Ye

Suppose we want to predict traffic using GNN

- Each node represents a sensor station recording the traffic speed. An edge connecting two nodes means these two sensor stations are connected on the road. The geographic diagram representing traffic speed of a region changes over time.

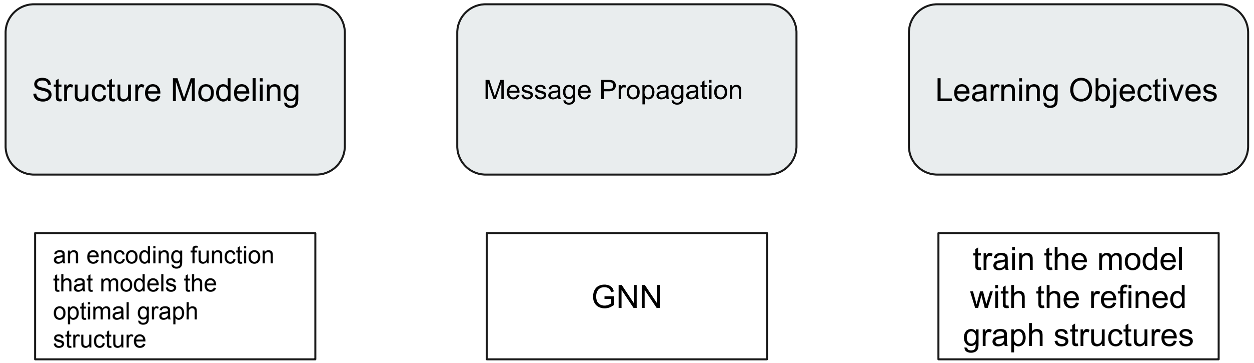




How do we solve a similar problem when graph structure is changing too?

Traditional Approaches on Structure Modeling

- Metric Learning Approaches
  - Edges are derived from learning a metric function between pairwise representations.
- Probabilistic Modeling Approaches
  - Assume the graph is generated via a sampling process from certain distributions and model the probability of sampling edges using learnable parameters.
- Direct Optimization Approaches
  - Refine the graph structure based on the original adjacency matrix by incorporating various graph priors.


Structure Modeling
Message Propagation
Learning Objectives
an encoding function that models the optimal graph structure
GNN
train the model with the refined graph structures

What are the problems with the traditional approaches?

- Need an assumption about the graph structure
- Lack Supervision

SLAPS

Self-Supervision Improves Structure Learning for Graph Neural Networks

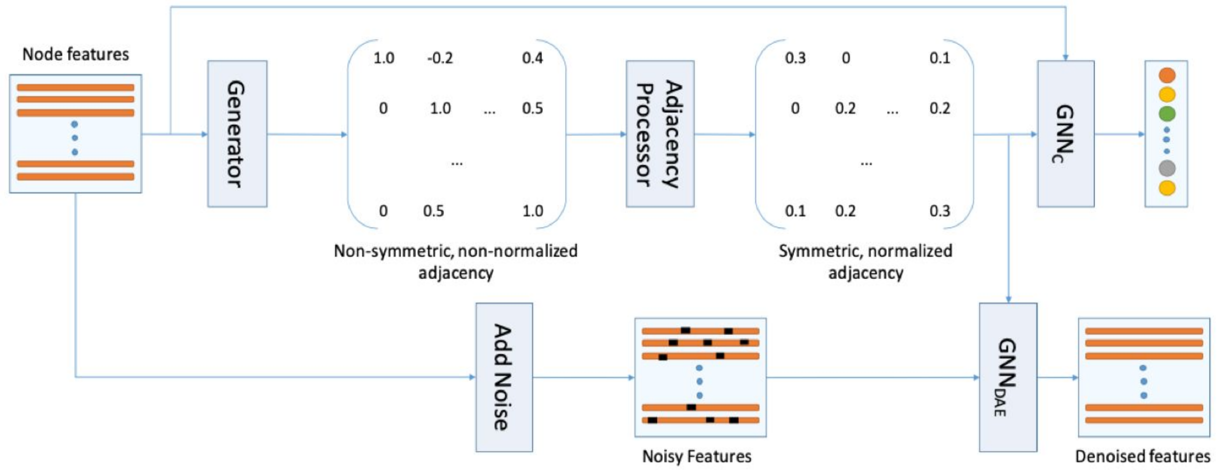


Figure 1: Overview of SLAPS. At the top, a generator receives the node features and produces a non-symmetric, non-normalized adjacency having (possibly) both positive and negative values (Section 4.1). The adjacency processor makes the values positive, symmetrizes and normalizes the adjacency (Section 4.2). The resulting adjacency and the node features go into GNN_C which predicts the node classes (Section 4.3). At the bottom, some noise is added to the node features. The resulting noisy features and the generated adjacency go into GNN_DAE which then denoises the features (Section 4.5).

Problem Formulation

What we know

Node Feature

T = 0,1, …, n-1

What we want to learn

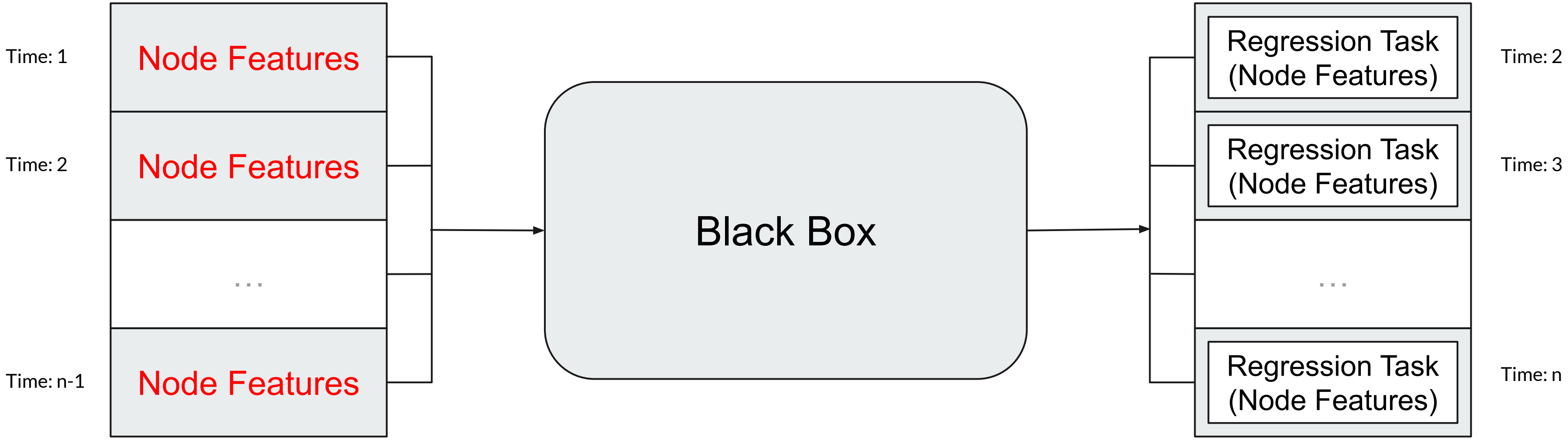
Latent Graph Structure

Node Feature

T = n

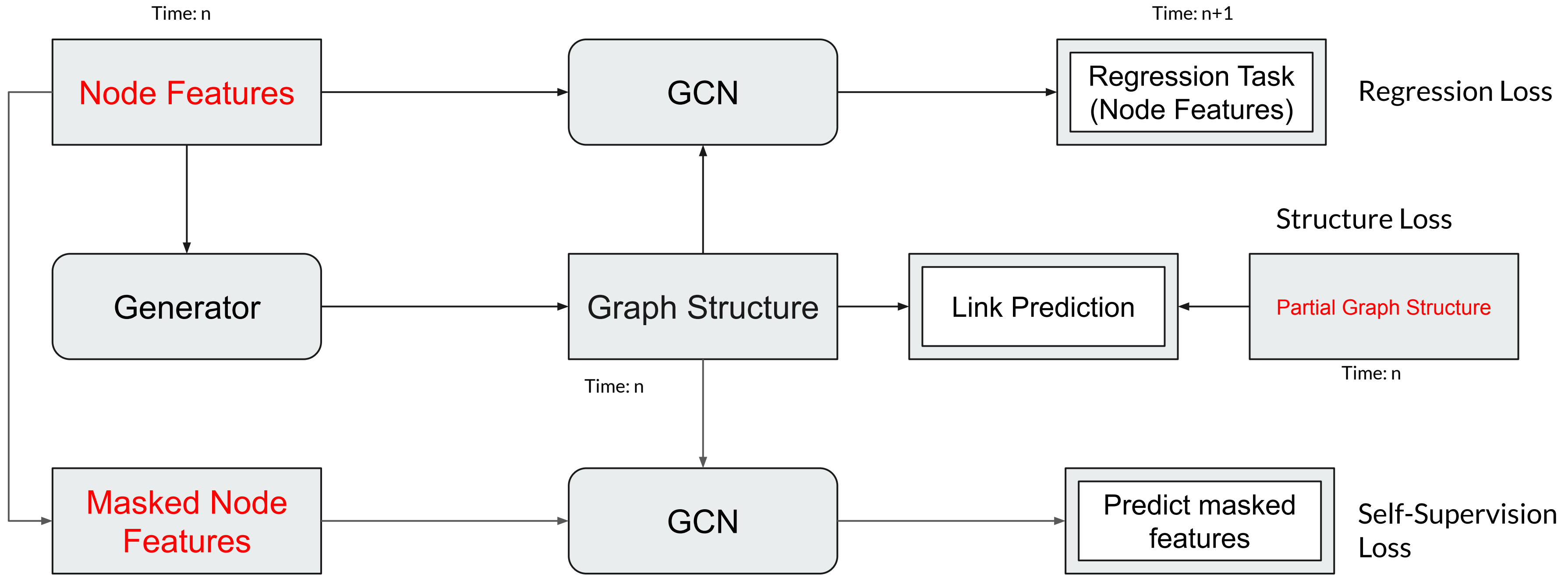
- Given Node Feature at time stamp T=0,...,T= n-1, we want to predict node feature at T = n.

Our Framework



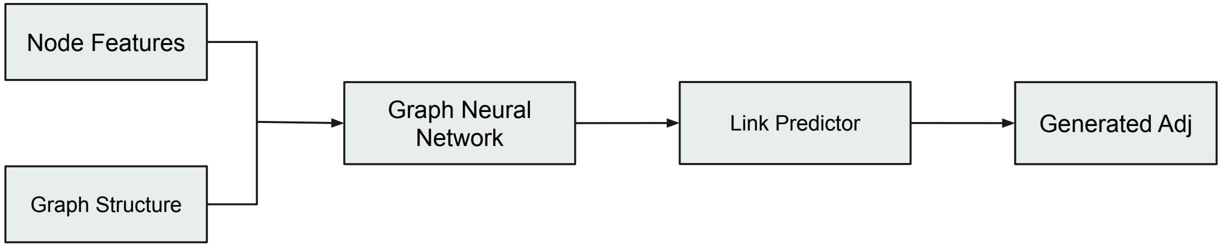
Let’s take a look at what’s inside the blackbox!

Our Framework

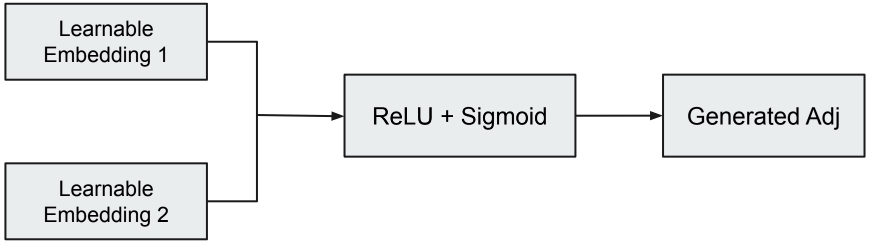


Generator

LP-Based



Self-adaptive Adj



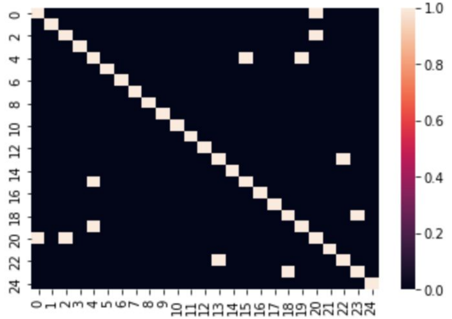
Dataset

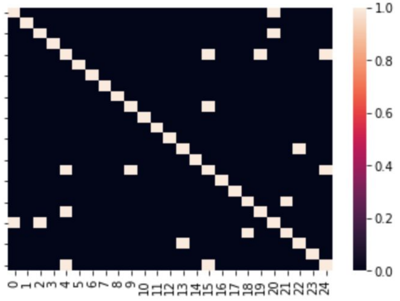
- We use a generated opinion migration dataset with different node sizes.
  - The social network data simulates the opinion migration of individuals in a social network over time
  - Node: social network users
  - Features: opinions (initialized with a uniform distribution in 2-d space)
  - We also set noise and sparsity parameters

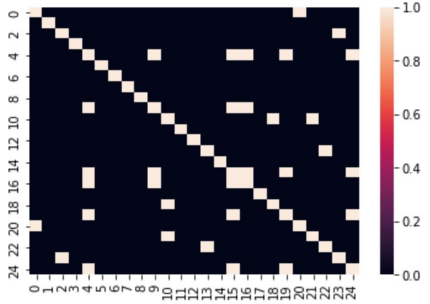
- We split the dataset into train, validation and test based on timespan.

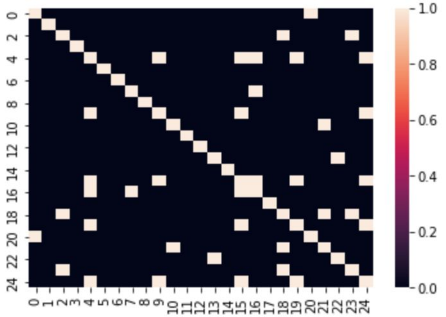
- Details about the dataset can be found in : Yupeng Gu, Yizhou Sun, and Jianxi Gao. 2017. The Co-Evolution Model for Social Network Evolving and Opinion Migration. In KDD'17.
- Details about the generation code can be found in: https://github.com/ZijieH/CG-ODE/blob/main/data/social/generate_socialNetwork.py

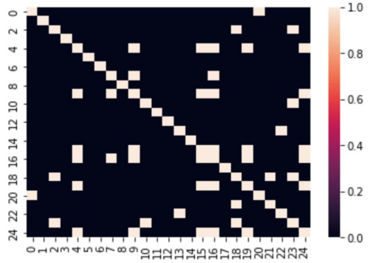
Visualization

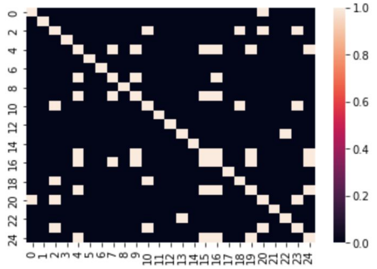

T = 0

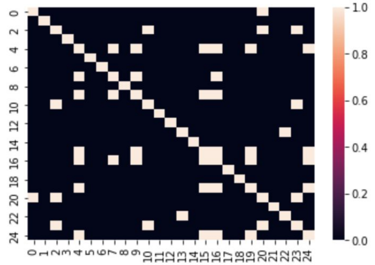

T = 1

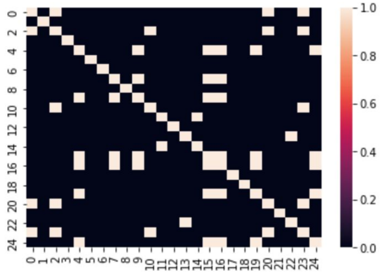

T = 2


T = 3


T = 4


T = 5


T = 6


T = 7

Experiment 1: structure accuracy

Acc = # correctly predicted edges / all edges

	Random structure + 400 epochs loss_regression	200 epochs loss_structure + 200 epochs loss_regression	400 epochs loss_regression	400 epochs (loss_regression+loss_structure)	200 epochs loss_structure + 200 epochs（loss_structure+loss_regression）
25 nodes	0.39	0.89	0.46	0.44	0.91
100 nodes	0.38	0.89	0.42	0.41	0.89
500 nodes	0.40	0.87	0.47	0.42	0.89

Experiment 2: downstream task

Predict the feature in next time stamp

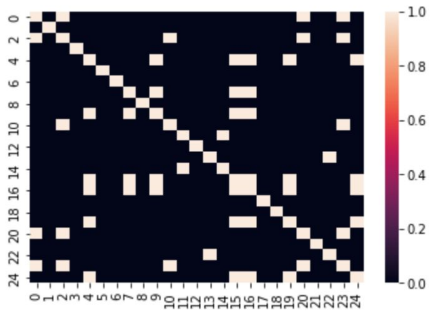
mse	Random structure + 400 epochs loss_regression	200 epochs loss_structure + 200 epochs loss_regression	400 epochs loss_regression	400 epochs (loss_regression+loss_structure)	200 epochs loss_structure + 200 epochs （loss_structure+loss_regression）
25 nodes	3.0699	0.0191	1.81	1.5856	0.6296
100 nodes	7.0415	2.54	2.57	2.6467	1.8911
500 nodes	3.18	1.69	2.71	1.7573	0.2873

Experiment 3: Dynamic Graph downstream task/graph generation(mse/acc) with Self Supervision

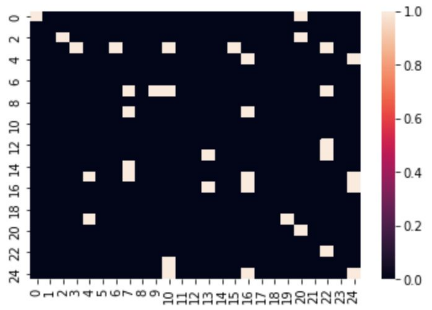
TimeSpan=10 Node num = 50 Percentage of Partial structure Ground truth(%)	Random Structure + 400 epochs Regression Loss	100 epochs Structure Loss + 300 epochs Regression Loss	100 epochs Structure Loss + 300 epochs (Regression Loss + Self-supervision 10% masked)	100 epochs Structure Loss + 300 epochs (Regression Loss + Self-supervision 20% masked)	100 epochs Structure Loss + 300 epochs (Regression Loss + Self-supervision 50% masked)
0%	20.98 / 33%	2.18 / 42%	2.91 / 43%	2.99 / 67%	3.6 / 53%
5%	-	2.91 / 49%	2.87 / 48%	2.70 / 55%	3.17 / 55%
20%	-	2.72 / 62%	2.73 / 55%	1.9 / 72%	2.8 / 81%
50%	-	2.72 / 64%	2.85 / 67%	1.62 / 81%	1.34 / 88%
100%	-	2.45 / 63%	2.32 / 72%	1.42 / 89%	1.32 / 92.6%

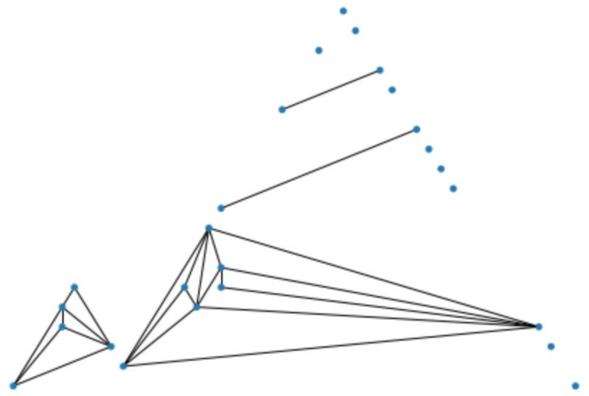
Sample Generated Structure

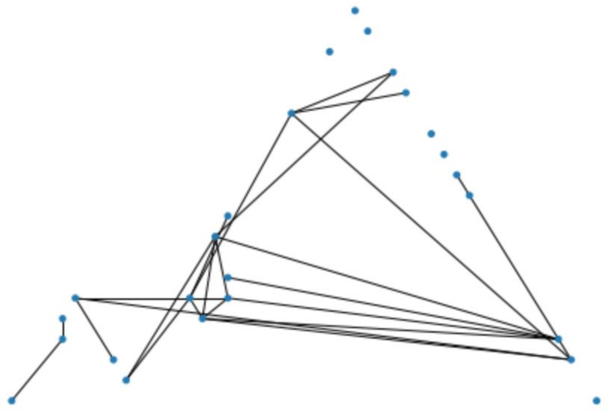
T = 8
Ground truth



T = 8
Generated







To Be Done

- Apply to Real World Datasets, like traffic datasets

- Adding Gumbel-Softmax as reparameterization trick

Gumbel Softmax

- Why use gumbel softmax?
  - Over-parameterized model
  - Discrete optimization

- Gumbel Softmax is a reparameterization trick used to generate differentiable discrete samples. Under appropriate hyper-parameter settings, Gumbel-Softmax is able to generate continuous vectors that are as “sharp” as one-hot vectors widely used to encode discrete data.

$$A_{ij} = \text{sigmoid}((\log(\theta_{ij}/(1-\theta_{ij})) + (g_{ij}^1 - g_{ij}^2))/s)$$

$g_{ij}^1, g_{ij}^2 \sim \text{Gumbel}(0,1)$ for all i, j

When the temperature $s \to 0$, $A_{ij} = 1$ with probability θ_{ij} and 0 with remaining probability.

Q & A