Gaussian-Process-Based Dynamic Embedding for Textual Networks

Pengyu Cheng, Yitong Li, Xinyuan Zhang, Liqun Chen, David Carlson, Lawrence Carin

Duke University

Graph Gaussian Process

We encode each $\boldsymbol{w}_n$ into the textual embedding space as $\boldsymbol{x}_n \in \mathbb{R}^d$.

We define a latent function $f(\boldsymbol{x})$ over textual embeddings with a GP prior $f(\boldsymbol{x}) \sim \mathcal{GP}(\mathbf{0}, k(\boldsymbol{x}_n, \boldsymbol{x}_{n'}))$.

To infer the structure embedding $\boldsymbol{s}_n$, we apply a graph diffusion on the top of the GP. At i-th dimension, the collection of structural embeddings follow

$$p(s_1^{(i)}, \ldots, s_N^{(i)} | \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) = \mathcal{N}(\mathbf{0}, \boldsymbol{P}_*^\intercal \boldsymbol{K}_{xx} \boldsymbol{P}_*),$$

where $\boldsymbol{P}_* = \Sigma_{j=0}^J \alpha_j \boldsymbol{P}^j$ includes different order graph topology information, $[\boldsymbol{K}_{xx}]_{ij} = k_\theta(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is a kernel matrix.

Inducing Points

GPs suffer from computational complexity with large data size N. To scale up the model, we use the inducing points.

Let $\boldsymbol{Z} = [\boldsymbol{z}_1, \cdots, \boldsymbol{z}_M]^\intercal$ with $M < N$ denote inducing points (pseudo-textual embeddings) in the same space with $\boldsymbol{X}$.

Assume $\boldsymbol{U} = [\boldsymbol{u}_1, \cdots, \boldsymbol{u}_M]^\intercal$ are corresponding the pseudo-structural embeddings of $\boldsymbol{Z}$, which is a function of $\boldsymbol{z}$ following the same GP function.

Given $\boldsymbol{Z}$ and $\boldsymbol{U}$, we have

$$p(\boldsymbol{S}_i | \boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{U}) = \mathcal{N}(\boldsymbol{\mu}_{S_i|Z}, \boldsymbol{\Sigma}_{S|Z}),$$
$$\boldsymbol{\mu}_{S_i|Z} = \boldsymbol{P}_*^\intercal \boldsymbol{K}_{XZ} (\boldsymbol{K}_{ZZ} + \sigma \boldsymbol{I}_M)^{-1} \boldsymbol{U}_i,$$
$$\boldsymbol{\Sigma}_{S|Z} = \boldsymbol{P}_*^\intercal \boldsymbol{K}_{XX} \boldsymbol{P}_* - \boldsymbol{P}_*^\intercal \boldsymbol{K}_{XZ} (\boldsymbol{K}_{ZZ} + \sigma \boldsymbol{I}_M)^{-1} \boldsymbol{K}_{ZX} \boldsymbol{P}_*,$$

where $[\boldsymbol{K}_{XZ}]_{nm} = k(\boldsymbol{x}_n, \boldsymbol{z}_m)$ and $[\boldsymbol{K}_{ZZ}]_{mm'} = k(\boldsymbol{z}_m, \boldsymbol{z}_{m'})$, $\boldsymbol{S}_i$ is the concatenation of the ith element from all node structural embeddings.

We use the mean $\hat{\boldsymbol{S}} = \boldsymbol{P}_*^\intercal \boldsymbol{K}_{XZ} (\boldsymbol{K}_{ZZ} + \sigma \boldsymbol{I}_M)^{-1} \boldsymbol{U}$, as unbiased estimation of structural embeddings.

When new nodes come, updates update

$$[\boldsymbol{s}, \boldsymbol{s}_{new}] = \Sigma_{j=0}^J \alpha_j \boldsymbol{P}_{new}^j \boldsymbol{K}_{x_{new}Z} (\boldsymbol{K}_{ZZ} + \sigma \boldsymbol{I}_M)^{-1} \boldsymbol{U}.$$

Illustration of DetGP

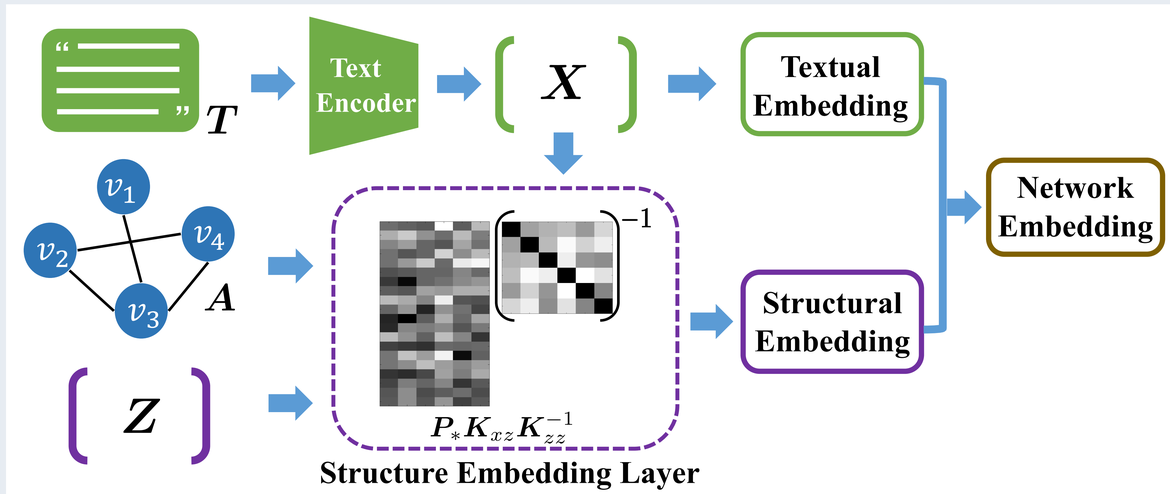


Figure: With connection information $\boldsymbol{A}$ of the network and textual side information $\boldsymbol{T} = \{\boldsymbol{t}_n\}_{n=1}^N$ as input, DetGP first encodes text $\boldsymbol{T}$ to a low-dimensional representation $\boldsymbol{X} = \{\boldsymbol{x}_n\}_{n=1}^N$, and then infers the structural embeddings by $\boldsymbol{X}$ and $\boldsymbol{A}$ via a Gaussian process. Inducing points $\boldsymbol{Z} = \{\boldsymbol{z}_m\}_{m=1}^M$ are used to reduce computational complexity. The output network embeddings combine the textual and structural embeddings.

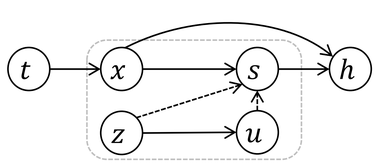
Background

Textual Networks widely appear in the real-word applications, *e.g.* citation networks.

The previous methods learn a **textual** embedding and a **structural** embedding for each document, and concatenate them together.

The **limitations** of these methods are:

- All nodes are required during training.
- When new nodes come or graph edges change, whole model needs to be re-trained to learn structural embeddings.



Notation

The Graph is given as $(\boldsymbol{W}, \boldsymbol{A})$, where $\boldsymbol{W} = \{\boldsymbol{w}_1 \ldots, \boldsymbol{w}_N\}$ is collection of text. $\boldsymbol{A}$ is the adjacency matrix. degree matrix $\boldsymbol{D} = \text{diag}(\boldsymbol{A}\mathbf{1}_N)$.
The normalized transition matrix

$$\boldsymbol{P} = (\boldsymbol{D} + \boldsymbol{I}_N)^{-1} (\boldsymbol{A} + \boldsymbol{I}_N)$$

Textual embeddings $\boldsymbol{X} = \{\boldsymbol{x}_n\}_{n=1}^N$.
Structural embeddings $\boldsymbol{S} = \{\boldsymbol{s}_n\}_{n=1}^N$.

Training Loss

We concatenate textual and structural embedding together as the node embedding $\boldsymbol{h}_i = [\boldsymbol{x}_i, \boldsymbol{s}_i]$
we minimize the negative sampling loss:

$$\mathcal{L} = -\frac{1}{|\mathcal{E}|} \sum_{(i,j) \in \mathcal{E}} \log \sigma(\boldsymbol{h}_i \cdot \boldsymbol{h}_j) + \frac{1}{N_s} \sum_{(i,j) \notin \mathcal{E}} \log \sigma(\boldsymbol{h}_i \cdot \boldsymbol{h}_j),$$

where $N_s = \#\{(i,j) \notin \mathcal{E}_t\}$ is the number of negative sample pairs.

Gaussian Process

A Gaussian Process (GP) $f(\boldsymbol{x})$ is a collection of random variables such that any subset of those variables are Gaussian distributed. Given $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$, $[f(\boldsymbol{x}_1), f(\boldsymbol{x}_2), \ldots, f(\boldsymbol{x}_n)]^\intercal \sim \mathcal{N}([m(\boldsymbol{x}_1), m(\boldsymbol{x}_2), \ldots, m(\boldsymbol{x}_n)]^\intercal, [k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{n \times n})$, where $m(\boldsymbol{x})$ is a mean function and $k(\cdot, \cdot)$ is a covariance kernel function.
To learn $y = f(\boldsymbol{x})$ with training data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ and unlabeled testing data $\{\boldsymbol{x}'_j\}_{j=1}^M$, $[f(\boldsymbol{x}_1), f(\boldsymbol{x}_2) \ldots, f(\boldsymbol{x}_n), f(\boldsymbol{x}'_1), \ldots, f(\boldsymbol{x}'_M)]$ follows a multivariate Gaussian distribution.
Given observations $f(\boldsymbol{x}_i) = y_i$, the conditional distribution for $[f(\boldsymbol{x}'_1), \ldots, f(\boldsymbol{x}'_M)]$ can be easily obtained, which is also Gaussian distributed.

Experiments

Static node classification:

	Cora				DBLP			
%Training Nodes	10%	30%	50%	70%	10%	30%	50%	70%
LINE	53.9	56.7	58.8	60.1	42.7	43.8	43.8	43.9
TADW	71.0	71.4	75.9	77.2	67.6	68.9	69.2	69.5
CANE	81.6	82.8	85.2	86.3	71.8	73.6	74.7	75.2
DMTE	81.8	83.9	86.3	87.9	72.9	74.3	75.5	76.1
WANE	81.9	83.9	86.4	88.1	NA	NA	NA	NA
DetGP (Wavg)	80.5	85.4	86.7	88.5	76.9	78.3	79.1	79.3
DetGP (DWavg)	**83.1**	**87.2**	**88.2**	**89.8**	**78.0**	**79.3**	**79.6**	**79.8**

Static link prediction:

	Cora					HepTh				
%Training Edges	15%	35%	55%	75%	95%	15%	35%	55%	75%	95%
node2vec	55.9	66.1	78.7	85.9	88.2	57.1	69.9	84.3	88.4	89.2
DeepWalk	56.0	70.2	80.1	85.3	90.3	55.2	70.0	81.3	87.6	88.0
CANE	86.8	92.2	94.6	95.6	97.7	90.0	92.0	94.2	95.4	96.3
DMTE	91.3	93.7	96.0	97.4	98.8	NA	NA	NA	NA	NA
WANE	91.7	94.1	96.2	**97.5**	**99.1**	92.3	95.7	97.5	97.7	**98.7**
DetGP (Wavg)	92.8	94.8	95.5	96.2	97.5	93.2	95.1	97.0	97.3	97.9
DetGP (DWavg)	**93.4**	**95.2**	**96.3**	**97.5**	98.8	**94.3**	**96.2**	**97.7**	**98.1**	98.5

Dynamic node classification:

	Cora				HepTh			
%Training Nodes	10%	30%	50%	70%	10%	30%	50%	70%
Only Text (Wavg)	61.2	77.9	87.9	90.3	68.3	83.7	84.2	86.9
Neighbor-Aggregate (Max-Pooling)	54.6	69.1	78.7	87.3	59.6	78.3	79.9	80.7
Neighbor-Aggregate (Mean)	61.8	78.4	88.0	91.2	68.2	83.9	85.5	88.3
GraphSAGE (Max-Pooling)	62.1	78.6	88.6	92.4	68.4	85.8	88.1	91.2
GraphSAGE (Mean)	62.2	79.1	88.9	92.6	69.1	85.9	89.0	92.4
DetGP	**62.9**	**81.1**	**90.9**	**93.0**	**70.7**	**86.6**	**90.7**	**93.3**

Dynamic link prediction:

	Cora				DBLP			
% Training Nodes	10%	30%	50%	70%	10%	30%	50%	70%
Only Text (Wavg)	60.2	76.3	83.5	84.8	56.7	67.9	70.4	73.5
Neighbor-Aggregate (Max-Pooling)	55.8	70.2	78.4	80.5	51.8	60.5	68.3	70.6
Neighbor-Aggregate (Mean)	60.1	77.2	84.1	85.0	56.8	68.2	71.3	74.7
GraphSAGE (Max-Pooling)	61.3	78.2	85.1	86.3	58.9	69.1	72.4	74.9
GraphSAGE (Mean)	61.4	78.4	85.5	**86.6**	59.0	69.3	72.7	75.1
DetGP	**62.1**	**79.3**	**85.8**	**86.6**	**60.2**	**70.1**	**73.2**	**75.8**