Advanced NLP Topic in Graph Embedding and Knowledge Graph

Jingbo Xia

Huazhong Agricultural University

xiajingbo.math@gmail.com

2020-02-17

Table of contents I

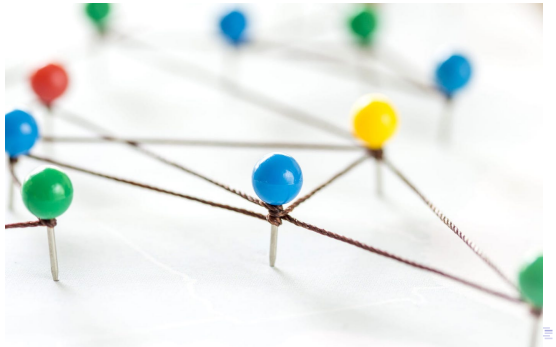
Outline

Graph Embedding and Knowledge Graph

Triple and knowledge graph (KG)

Fundamentals in KG:

1. $\mathbb{E}$: Set of entities
2. $\mathbb{R}$: Set of relations
3. $\mathbb{D}+ = (h, r, t)$: Set of facts (triples) where
   - head entity $h \in \mathbb{E}$.
   - tail entity $t \in \mathbb{E}$
   - relation $r \in \mathbb{R}$ between head and tail



Graph Embedding and Knowledge Graph I

Overview

Characteristics:

1. Availability of huge data
2. Use both graphs and texts

Knowledge graph applications

1. Link prediction: $(?, r, t)$ or $(h, ?, t)$ or $(h, r, ?)$
2. Triple classification: $(h, r, t)?$
3. Entity classification: $(h, IsA, ?)$
4. Entity resolution: $(x, EqualTo, y)$
5. Relation discovery in bio-field: $(?Protein, Targetof, ?chemical)$

Text-related applications

1. Relation extraction
   - Sentence: "Chemical targeted on Protein"
   - Detected entities: $h = Protein, t = Chemical : (h, TargetOf, t)$.
2. Question answering

Graph Embedding and Knowledge Graph II

Overview

1. Question: "What targeted on protein?"
2. Supporting fact: $(Protein, Targetof, Chemical)$
3. Answer: Chemical

Embedding tricks

1. Embed the entities as vectors
2. Embed the relations as
   - matrices
   - tensors
   - multivariate Gaussian distributions

Knowledge inference

1. Define a scoring function $f_r(h, t)$ on each fact (h, r, t) to measure plausibility of fact
2. Maximize total plausibility of facts in knowledge graph $\mathbb{D}+$

One roughly categorizes such embedding techniques into two groups [1]:

Graph Embedding and Knowledge Graph III

Overview

1. Translational distance models
   TransE, TransH, TransR, TransD, TranSparse, TransM, ManifoldE, TransF, TransA, KG2E, TransG, UM, SE, RotatE.
2. Semantic matching models
   RESCAL, TATEC, DistMult, HolE, ComplEX, ANALOGY, SME, NTN, SLM, MLP, NAM, SimplE, TuckER, ConvE, ConvKB, CapsE, SACN.

The former use distance-based scoring functions, and the latter is similarity-based ones.

Outline

Translational Distance Models I

TransE

Trans E, a Translational Distance model [3].

1. Relation = vector r
2. Apply relation = translate head entity
   - Add vector of relation r to head entity h to obtain tail entity t
   - $h + r \approx t$
   - (Similar motive to Word2Vec) Japan−Tokyo=China−Beijing
3. Minimize distance with actual tail entity (using $L1$ or $L2$ norm)
   - $f_r(h,t) = -||h + r - t||_1$
   - $f_r(h,t) = -||h + r - t||_2$
4. Limitation
   - Models a relation as a function
   - Does not work for $1:n$, $n:1$, $n:n$ relations

Translational Distance Models II

TransE

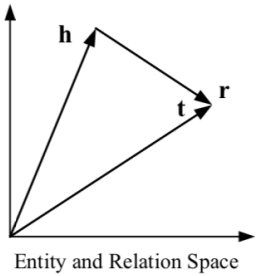


图 3: TransE

This figure visualized the idea of TransE.

[3] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Proc. Workshop at ICLR' 13, 2013a.

Translational Distance Models I

From TransE to TransH and TransR

TransH [4].

1. For a relation r, position the relation-specific translation vector d_r in the relation-specific hyperplane, $w_r x + b = 0$ with w_r (the normal vector，法向量), rather than in the same space of entity embeddings.
2. Project both head and tail entities onto the hyperplane, too.
3. Minimize distance with projected head and tail

$$f_r(h,t) = ||h_\perp + d_r - t_\perp|| = ||(h - w_r^T h w_r) + d_r - (t - w_r^T t w_r)||.$$

4. It enables different roles of an entity in different relations/triplets.
5. The score is expected to be lower for a golden triplet and higher for an incorrect triplet.
6. The model parameters are, all the entities' embeddings, $e_i{}_{i=1}^{|E|}$, all the relations' hyperplanes and translation vectors, $(w_r, d_r)_{r=1}^{|R|}$.

Translational Distance Models II

From TransE to TransH and TransR

7. TransH uses the following margin-based ranking loss:

$$\mathcal{L} = \sum_{(h,r,t)\in\Delta} \sum_{(h',r',t')\in\Delta'_{(h,r,t)}} [f_r(h,t) + \gamma - f_r(h',t')]_+,$$

   where $[x]_+ = \max(0,x)$, Δ is the set of positive (golden) triplets, $\Delta_{(h,r,t)}$ denotes the set of negative triplets constructed by corrupting (h,r,t), γ is the margin separating positive and negative triplets.
8. More constraints:
   - $\forall e \in E, ||e|| \le 1$, //scale
   - $\forall r \in R, \frac{|w_r^T d r|}{||dr||_2} \le \varepsilon$, //orthogonal
   - $\forall r \in R, ||w_r||_2 = 1$, //unit normal vector

Translational Distance Models III

From TransE to TransH and TransR

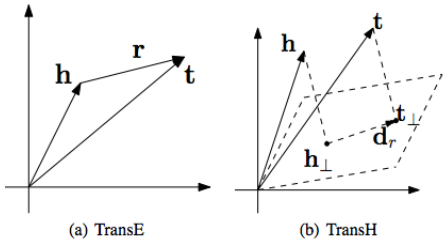


图 4: TransE and From TransE to TransH and TransR

[4]Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph and text jointly embedding. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1591-1601.
http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/download/8531/8546

Translational Distance Models II

From TransE to TransH and TransR

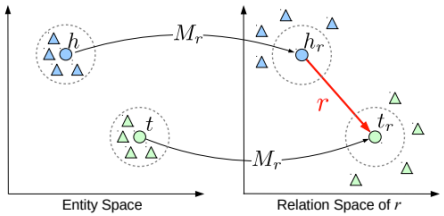


图 5: From TransE to TransH and TransR

[5]Lin H , Liu Y , Wang W , et al. Learning Entity and Relation Embeddings for Knowledge Resolution[J]. Procedia Computer Science, 2017, 108:345-354.
http://nlp.csai.tsinghua.edu.cn/~lyk/publications/aaai2015_transr.pdf

Translational Distance Models I

From TransE to TransH and TransR

TransR [5]

1. In TransR, for each triple (h, r, t), entities embeddings are set as $h, t \in \mathbb{R}^k$ and relation embedding is set as $r \in \mathbb{R}$.
2. For each relation r, we set a projection matrix $M_r \in \mathbb{R}^{k \times d}$, which may projects entities from entity space to relation space.
3. Define the projected vectors of entities as$h_r = hM_r$, $t_r = tM_r$.
4. The score function is correspondingly defined as

$$f_r(h, t) = ||h_r + r - t_r||.$$

5. More constraints: $||h||_2 \leq 1, ||r||_2 \leq 1, ||t||_2 \leq 1, ||hM_r||_2 \leq 1, ||tM_r||_2 \leq 1.$

Outline

RESCAL [6].
Relations are modeled as triples of the form (subject, predicate, object).
A tensor entry $\mathcal{X}_{ijk} = 1$ denotes the fact that there exists a relation (i-th entity, k-th predicate, j-th entity).

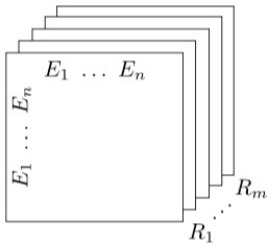


图 6: Tensor model for relational data. $E_1, \cdots, E_n$ denote the entities, while $R_1, \cdots, R_m$ denote the relations in the domain.

[6]M. Nickel, V. Tresp, and H.-P. Kriegel, "A three-way model for collective learning on multi-relational data," in Proc. 28th Int. Conf. Mach. Learn., 2011, pp. 809–

More precisely, they employed the following rank-r factorization, where each slice $\mathcal{X}_k$ is factorized as

$$\mathcal{X}_k \sim A\mathcal{R}_k A^T, \text{ for } k = 1, \cdots, m. \quad (1)$$

Here, $A = (A_1, \cdots, A_r)$ is a $n \times d$ matrix that contains the latent-component representation of the entities in the domain[7] and R_k is an asymmetric $d \times d$ matrix that models the interactions of the latent components in the k-th predicate.

The factor matrices A and R_k can be computed by solving the regularized minimization problem:

$$\min_{A,R_k} f(A, R_k) + g(A, R_k)$$

where

$$f(A, R_k) = \frac{1}{2}(\sum_k ||\mathcal{X}_k - AR_kA^T||_F^2), \quad g(A, R_k) = \frac{1}{2}\lambda(||A||_F^2 + \sum_k ||R_k||_F^2).$$

[7]I have similar analysis on formula (1) later.

An important aspect of RESCAL for collective learning and what distinguishes it from other tensor factorizations like CP is that the entities of the domain have a unique latent-component representation, regardless of their occurrence as subjects or objects in a relation, as they are represented both times by the matrix A.
The effect of this modeling becomes more apparent by looking at the element-wise formulation of the equation, namely

$$f(A, R_k) = \frac{1}{2}(\mathcal{X}_{ijk} - A_i R_k A_j^T)^2.$$

Here, A_i and A_j denote the i-th and j-th row of A and thus are the latent-compnent representation of the i-th and j-th entity.
By holding A_j and R_k fixed, it is clear that the latent-component representation A_i depends on A_j as well as the existence of the triple (i-th entity, k-th predicate, j-th entity) represented by $\mathcal{X}_{ijk}$.

The latent-component representations of Al and $Lyndon$ will be similar to each other in this example, as both representations reflect that their corresponding entities are related to the object $PartyX$. Because of this, $Bill$ and $John$ will also have similar latent-component representations. Consequently, the product $A_{Bill}R_{party}A_{PartyX}^T$ will yield a similar value to $A_{John}R_{party}A_{PartyX}^T$ and as such the missing relation can be predicted correctly.

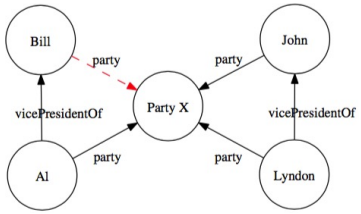


图 7: Visualization of a subgraph of the relational graph for the US presidents example. The relation marked red is unknown.

Semantic Matching Models

Tensor decomposition (张量分解) and RESCAL

An interesting view of formula (1) Let's recall Tucker decomposition: For a 3-way tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ with $\mathcal{G} \in \mathbb{R}^{P \times Q \times R}$, $A \in \mathbb{R}^{I\ timesP}$, $B \in \mathbb{R}^{J\ timesQ}$, $C \in \mathbb{R}^{I\ timesR}$ as follows:

$$\begin{aligned} \min_{\mathcal{X}} ||\mathcal{X} - \hat{\mathcal{X}}|| \text{ with } \quad \hat{\mathcal{X}} &= \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} a_r \circledcirc b_r \circledcirc c_r \\ &= \mathcal{G} \times_1 A \times_2 B \times_3 C \\ &= [[\mathcal{G}; A, B, C]] \end{aligned} \tag{2}$$

The matricized version of the above version is:

$$\begin{aligned} \hat{\mathcal{X}_{(1)}} &= A G_{(1)} (C \otimes B)^T, \\ \hat{\mathcal{X}_{(2)}} &= B G_{(1)} (C \otimes A)^T, \\ \hat{\mathcal{X}_{(3)}} &= C G_{(1)} (B \otimes A)^T. \end{aligned} \tag{3}$$

Note: $\mathcal{X}_k \sim A \mathcal{R}_k A^T$ (formula (1)) can be regarded a restricted version of Tucker decomposition. Let $\mathcal{X}_{(n)} = A G_{(n)} (C \otimes B)^T$ be the matricized form of the Tucker3 decomposition of $\mathcal{X}$, the formula (1), holds, when the factors B and C are constrained to $B = A$ and $C = I_k$, while G is holding the slices R_k .

Note: Interestingly, my analysis led the above tensor decomposition to a

Semantic Matching Models I

From RESCAL to DistMult and complEX

Relation embedding of RESCAL

For easy comparison of RESCAL, DistMult, and complEX, one defines the score of a fact (h, r, t) in RESCAL by a bilinear function

$$f_r(h, t) = \boldsymbol{h}^T \mathcal{R}_r \boldsymbol{t},$$

where $\boldsymbol{h}, \boldsymbol{t} \in \mathbb{R}^d$ are vector representations of the entities, and $\mathcal{R}_r \in \mathbb{R}^{d \times d}$ is a matrix associated with the relation.

Variant of RESCAL: DistMult [8]

Meanwhile, DistMult simplifies RESCAL by restricting $\mathcal{R}_r$ to diagonal matrices. For each relation r, it introduces a vector embedding $\boldsymbol{r} \in \mathbb{R}^d$ and requires $\mathcal{R}_r = diag(\boldsymbol{r})$. The scoring function is hence defined as

$$f_r(h, t) = \boldsymbol{h}^T diag(\boldsymbol{r}) \boldsymbol{t}$$

.

Semantic Matching Models II

From RESCAL to DistMult and complEX

Variant of RESCAL: ComplEx [9]

Moreover, ComplEx extends DistMult by introducing complex-valued embeddings so as to better model asymmetric relations. In ComplEx, entity and relation embeddings h, r, t no longer lie in a real space but a complex space, say $\mathbb{C}^d$. The score of a fact (h, r, t) is defined as

$$f_r(h, t) = Re(\boldsymbol{h}^T diag(\boldsymbol{r}) \bar{\boldsymbol{t}}),$$

where $\bar{t}$ is the conjugate of t and $Re(\cdot)$ means taking the real part of a complex value.

[8] Yang, B., Yih, W.t., He, X., Gao, J., and Deng, L.: Embedding entities and relations for learning and inference in knowledge bases. arXiv preprint arXiv:1412.6575 (2014)

[9] Trouillon,T.,Welbl,J.,Riedel,S.,Gaussier,E´.,andBouchard,G.:Comple xembeddings for simple link prediction. ICML. pp. 2071-2080 (2016), http://www.jmlr.org/proceedings/papers/v48/trouillon16.pdf

Outline

Neural Network on Graphs I

From CNN, Graph embedding to GNN

Graph neural networks (GNNs) [10] are connectionist models that capture the dependence of graphs via message passing between the nodes of graphs. Unlike standard neural networks, graph neural networks retain a state that can represent information from its neighborhood with arbitrary depth.
Typical GNN methods: ChebNet, GCN, DCNN, GraphSAGE, GAT, GGNN, Tree LSTM.

1. **Why Graph?** Graphs can be used as denotation of a large number of systems across various areas including social science (social networks), natural science (physical systems and protein-protein interaction networks), knowledge graphs and many other research areas.
2. **What for?** Node classification, link prediction, and clustering
3. **Started from CNN** The first motivation of GNNs roots in convolutional neural networks (CNNs).
   - The keys of CNNs: local connection, shared weights and the use of multi-layer.

Neural Network on Graphs II

From CNN, Graph embedding to GNN

- However, CNNs can only operate on regular Euclidean data like images (2D grid) and text (1D sequence) while these data structures can be regarded as instances of graphs.

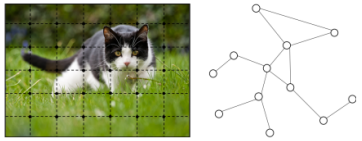


图 8: Image in Euclidean space, and graph in non-Euclidean space

- Neither CNNs nor RNNs handles the graph input properly as it stacks the feature of nodes by a specific order. However, there is not a natural order of nodes in the graph.

4. **Combinded with graph embedding**, which learns to represent graph nodes, edges or subgraphs in low-dimensional vectors.

Neural Network on Graphs III

From CNN, Graph embedding to GNN

- In the field of graph analysis, traditional machine learning approaches usually rely on hand engineered features and are limited by its inflexibility and high cost.
- Following the idea of representation learning and the success of word embedding, **DeepWalk**, which is regarded as the first graph embedding method, applies SkipGram model on the generated random walks.
- More progress: **node2vec, LINE, TADW**.
- Drawbacks. First, no parameters are shared between nodes in the encoder; Second, the direct embedding methods lack the ability of generalization, cannot deal with dynamic graphs or generalize to new graphs.

5. Here comes GNN. Based on CNNs and graph embedding, graph neural networks (GNNs) are proposed to
   - collectively aggregate information from graph structure.
   - model input and/or output consisting of elements and their dependency.
   - simultaneously model the diffusion process on the graph with the RNN kernel.

[10] Zhou, Jie , et al. "Graph Neural Networks: A Review of Methods and Applications." (2018). https://arxiv.org/abs/1812.08434

Banach fixed point (不动点) theorem/ contract map (压缩映射) theorem I

定理 (Banach fixed point theorem -Banach 不动点理论))

Let $\mathcal{K}$ be a complete metric space in which the distance between two points P and Q is denoted $d(P, Q)$.
And let $F : \mathcal{K} \longrightarrow \mathcal{K}$ be a contraction map (压缩映射), i.e., there exists $c \in (0, 1)$ such that for all $P, Q \in \mathcal{K}$, then $d(F(P), F(Q)) \leq c \cdot d(P, Q)$.

Then F has a unique fixed point, i.e. there exists a unique $x \in \mathcal{K}$ such that

$$F(x) = x.$$

In short,
"从完备度量空间 $(\mathcal{K}, d)$到自身的压缩映射有唯一的不动点。"

Refers to here [11].

Banach fixed point (不动点) theorem/ contract map (压缩映射) theorem II

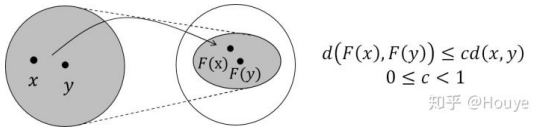


图 9: Description of contract map (压缩映射)

Ex.

Assume $A = (a_{i,j})_{n\times n}$, $x = (x_1, x_2, \cdots, x_n)^T \in \mathbb{R}^n$, $y = (y_1, y_2, \cdots, y_n)^T \in \mathbb{R}^n$. For each i $(1 \le i \le n)$, matrix A satisfies $\sum_{i,j=1}^{n} |a_{i,j}| < 1$, then linear equation system $Ax + b = x$ has an unique solution.

Banach fixed point (不动点) theorem/ contract map (压缩映射) theorem III

Ex.

Assume $A = (a_{i,j})_{n\times n}$, $x = (x_1, x_2, \cdots, x_n)^T \in \mathbb{R}^n$, $y = (y_1, y_2, \cdots, y_n)^T \in \mathbb{R}^n$. For each i $(1 \le i \le n)$, matrix A satisfies $\sum_{i,j=1}^{n} |a_{i,j}| < 1$, then linear equation system $Ax + b = x$ has an unique solution.

Solution: Define distance

$$d(x, y) = \max_{1\le i\le n} \{|x_i - y_i|\}.$$

It is easy to prove $\mathbb{R}^n, d$ is a metric space. So we define a map T,

$$T(x) = Ax + b.$$

Banach fixed point (不动点) theorem/ contract map (压缩映射) theorem IV

Then, we know

$$\begin{aligned} d(T(x), T(y)) &= \max_{1\le i\le n} \{|\sum_{j=1}^{n} a_{ij}(x_j - y_j)|\} \\ &\le \max_{1\le i\le n} \{\sum_{j=1}^{n} |a_{ij}|\} \cdot \max_{1\le j\le n} \{|(x_j - y_j)|\} \\ &= c \cdot d(x, y), \end{aligned}$$

here $c = \max_{1\le i\le n} \{|(x_j - y_j)|\} \le 1$.

So, from Banach fixed point theorem, we know that the system has a unique solution.

[11] http://dmuw.zum.de/images/b/bd/Banach2.pdf

Neural Network on Graphs I

GNN

GNN is first proposed in the paper by F. Scarselli et al.[12]

1. GNNs propagate on each node v respectively, ignoring the input order.
2. The output of GNNs is invariant for the input order of nodes.
3. In a graph, each node is naturally defined by its features $\boldsymbol{x}_v$ and the related nodes. The target of GNN is to learn a state embedding $\boldsymbol{h}_v \in \mathbb{R}^s$ which contains the information of neighborhood for each node v!!!
4. The state embedding $\boldsymbol{h}_v$ is an s-dimension vector of node v and can be used to produce an output $\boldsymbol{o}_v$ such as the node label.
5. Notations:
   - $\boldsymbol{H}$, all the state embeddings,
   - $\boldsymbol{O}$, all the output embeddings,
   - $\boldsymbol{X}$, all the features.

Neural Network on Graphs II

GNN

6. f, local transition function, is shared among all nodes and updates the node state $\boldsymbol{H}$ according to the input neighborhood $ne[v]$.

$$\boldsymbol{h}_v = f(\boldsymbol{x}_v, \boldsymbol{x}_{co[v]}, \boldsymbol{h}_{ne[v]}, \boldsymbol{x}_{ne[v]}),$$

here

- $\boldsymbol{x}_v$, features of v;
- $\boldsymbol{x}_{co[v]}$, the features of v's edges;
- $\boldsymbol{h}_{ne[v]}$, the states of the nodes in the neighborhood of v;
- $\boldsymbol{x}_{ne[v]}$, the features of the nodes in the neighborhood of v.

f is learned from a neural network.

Neural Network on Graphs III

GNN

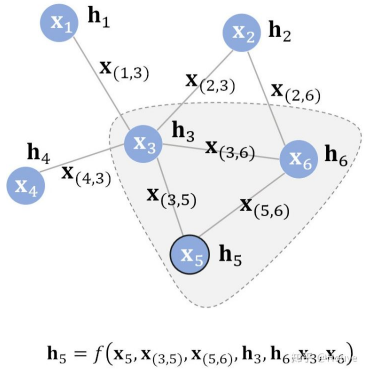


图 10: h_v embedding example in GNN

The figure [13] well depicts an example of f.

Neural Network on Graphs IV

GNN

7. Actually, f is learned by a Feed-forward Neural Network (FNN). So, if we make FNN a contract map, we can have a unique solution by:

$$\boldsymbol{h}_v^{t+1} = f(\boldsymbol{x}_v, \boldsymbol{x}_{co[v]}, \boldsymbol{h}_{ne[v]}^t, \boldsymbol{x}_{ne[v]}) = \sum_{u \in ne(v)} FNN([\boldsymbol{x}_v; \boldsymbol{x}_{(u,v)}; \boldsymbol{h}_u^t; \boldsymbol{x}_u]). \quad (4)$$

8. **How to let FNN be contract map?**
Please consider that if $||F(x) - F(y)|| \leq c \cdot ||x - y||$, $0 \leq c < 1$, then $\frac{||F(x)-F(y)||}{||x-y||} \leq c$. So, $\frac{||F(x)-F(x-\Delta x)||}{||\Delta x||} \leq c$, it suffies to know $||F'(x) \leq c||$. By reversing the idea, GNN set up the following loss function:

$$\begin{aligned}\mathcal{J} &= Loss + \lambda \cdot \max(\frac{||\partial FNN||}{||\partial \boldsymbol{h}||} - c, 0) \\ &= Loss + \lambda \cdot RELU(\frac{||\partial FNN||}{||\partial \boldsymbol{h}||} - c)\end{aligned}$$

where $c \in (0, 1)$ and λ is hyper parameter.

Neural Network on Graphs V

GNN

9. **Define Loss!** Use some downstream evaluation with output $\boldsymbol{o}_v$.

$$Loss = \sum_{i=1}^{p} (\boldsymbol{t}_i - \boldsymbol{o}_i),$$

where p is the number of supervised nodes, $\boldsymbol{t}_v$ is the target infromation for the supervision.

10. **How to obtain $\boldsymbol{o}_v$ with a g, which is also implemented with a FNN.**
Let g be the local output function that describes how the output is produced.

$$\boldsymbol{o}_v = g(\boldsymbol{h}_v, \boldsymbol{x}_v).$$

11. Note that the computations described in f and g can be interpreted as the feedforward neural networks.

Neural Network on Graphs VI

GNN

12. Compact form as:

$$\begin{aligned} \boldsymbol{H} &= F(\boldsymbol{H}, \boldsymbol{X}), \\ \boldsymbol{O} &= G(\boldsymbol{H}, \boldsymbol{X}_N), \end{aligned} \tag{5}$$

where F, the global transition function, and G, the global output function are stacked versions of f and g for all nodes in a graph.

13. Iteration scheme:

$$\boldsymbol{H}^{t+1} = F(\boldsymbol{H}^t, \boldsymbol{X}).$$

The states $\boldsymbol{h}_v^t$ are iteratively updated by Eq. (4) until a time T. They approach the fixed point solution of Eq. (5): $H(T) \approx H$.

14. And finally here comes up with the structure of GNN:

Neural Network on Graphs VII

GNN

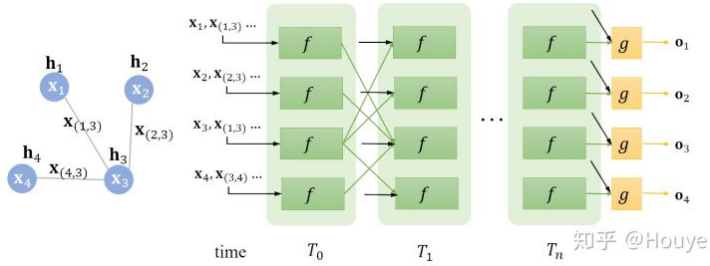


图 11: Structure of GNN

$t = T_0, T_1, \cdots, T_n$ is for iteration.

[12] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," IEEE TNN 2009, vol. 20, no. 1, pp. 61–80, 2009. http://persagen.com/files/misc/scarselli2009graph.pdf

[13] https://zhuanlan.zhihu.com/p/108294485

Neural Network on Graphs III

Spectral signal on graph, from Laplacian operator to Fourier transform

Definition of Laplacian matrix [14]:
Given a simple graph G with n vertices, its Laplacian matrix $L_{n\times n}$ is defined as [15]:

$$L = D - A, \tag{6}$$

where D is the degree matrix [16] and A is the adjacency matrix [17] of G.

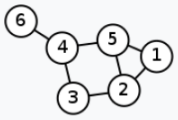
Labelled graph	Degree matrix	Adjacency matrix	Laplacian matrix
	$\begin{pmatrix} 2&0&0&0&0&0\\ 0&3&0&0&0&0\\ 0&0&2&0&0&0\\ 0&0&0&3&0&0\\ 0&0&0&0&3&0\\ 0&0&0&0&0&1 \end{pmatrix}$	$\begin{pmatrix} 0&1&0&0&1&0\\ 1&0&1&0&1&0\\ 0&1&0&1&0&0\\ 0&0&1&0&1&1\\ 1&1&0&1&0&0\\ 0&0&0&1&0&0 \end{pmatrix}$	$\begin{pmatrix} 2&-1&0&0&-1&0\\ -1&3&-1&0&-1&0\\ 0&-1&2&-1&0&0\\ 0&0&-1&3&-1&-1\\ -1&-1&0&-1&3&0\\ 0&0&0&-1&0&1 \end{pmatrix}$

图 12: Laplacian matrix

Neural Network on Graphs IV

Spectral signal on graph, from Laplacian operator to Fourier transform

Assume $f = (f_1, f_2, \cdots, f_n)^T$ is a representation of the features of the nodes in graph. We have the following result:

定理 ()

$\forall f \in \mathbb{R}^n$, *we have*

$$f^T L f = \frac{1}{2}\sum_{i,j=1}^{n} A_{ij}(f_i - f_j)^2.$$

Proof:

$$\begin{aligned} f^T L f &= f^T D f - f^T A f \\ &= \sum_{i=1}^{n} d_{ii} f_i^2 - \sum_{i,j=1}^{n} A_{ii} f_i f_j \\ &= \tfrac{1}{2}\Big(\sum_{i=1}^{n} d_{ii} f_i^2 - 2\sum_{i,j=1}^{n} A_{ii} f_i f_j + \sum_{i=1}^{n} d_{ii} f_i^2\Big) \\ &= \frac{1}{2}\sum_{i,j=1}^{n} A_{ij}(f_i - f_j)^2. \end{aligned}$$

Neural Network on Graphs V

Spectral signal on graph, from Laplacian operator to Fourier transform

Symmetric normalized Laplacian (对称正规拉普拉斯矩阵)
The symmetric normalized Laplacian matrix is defined as:

$$L^{\text{sym}} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}. \tag{7}$$

定理 (Normalized Laplacian matrix result)

The elements of L^{sym} are given by

$$L^{sym}_{i,j} = \begin{cases} 1 & \text{if } i = j \text{ and } \deg(v_i) \neq 0 \\ -\frac{1}{\sqrt{\deg(v_i)\deg(v_j)}} & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise.} \end{cases}$$

Note: Please compute the L^{sym} of the graph in figure 12.
Understand how the normalization of the Laplacian matrix look like.

Neural Network on Graphs VI

Spectral signal on graph, from Laplacian operator to Fourier transform

Please also note that we have another presentation of this result:

定理

$$L^{sym} = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}.$$

Here, $I_N \in \mathbb{R}^n$ is the identity matrix.
Since the degree matrix D is a diagonal matrix with positive diagonal elements, the above result is straightforward to obtain.

Neural Network on Graphs VII

Spectral signal on graph, from Laplacian operator to Fourier transform

Spectral decomposition of Laplacian matrix.

定理

$$L = U \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix} U^{-1}, \text{ or, } L = U \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix} U^T,$$

where $U = (u_1, u_2, \cdots, u_n)$, u_i is the eigenvector of L.

Taking into account the Schmidt orthogonal diagonalization of the symmetric real matrix L, this result is straightfoward.

Neural Network on Graphs VIII

Spectral signal on graph, from Laplacian operator to Fourier transform

Gradient ∇, divergence $\nabla\cdot$ and Laplacian operator Δ
Here is the definition of gradient, divergence, and Laplacian operator defined on the 3D Euclidean space $\mathbb{R}^3$. Please note that this definition is surely generalized to a Euclidean space $\mathbb{R}^n$ with arbitrary n...

$$\begin{aligned} &\text{(1) Gradient (梯度): } \nabla f = \frac{\partial f}{\partial x}\mathbf{i} + \frac{\partial f}{\partial y}\mathbf{j} + \frac{\partial f}{\partial z}\mathbf{k} = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{pmatrix}. \\ &\text{(2) Divergence (散度): } \nabla \cdot \boldsymbol{F} = \frac{\partial \boldsymbol{F}_x}{\partial x} + \frac{\partial \boldsymbol{F}_y}{\partial y} + \frac{\partial \boldsymbol{F}_z}{\partial z} \\ &\text{(3) Laplacian operator: } \Delta f = \nabla^2 f = \nabla \cdot \nabla f = div(gradf). \end{aligned} \tag{8}$$

These are very important things to mention!

1. Gradient (∇) works on a multi variate function, and obtains a vector.
2. Divergence ($\nabla\cdot$) works on a vector, and obtain a real value.
3. Laplacian operator (Δ) is the **Divergence of Gradient !!!**

Neural Network on Graphs IX

Spectral signal on graph, from Laplacian operator to Fourier transform

In mathematics, the gradient is a multi-variable generalization of the derivative. While a derivative can be defined on functions of a single variable, for functions of several variables, the gradient takes its place. The gradient is a vector-valued function, as opposed to a derivative, which is scalar-valued.

Consider a room in which the temperature is given by a scalar field, T, so at each point (x, y, z) the temperature is $T(x, y, z)$. (Assume that the temperature does not change over time.) At each point in the room, the gradient of T at that point will show the direction in which the temperature rises most quickly. The magnitude of the gradient will determine how fast the temperature rises in that direction.

Neural Network on Graphs X

Spectral signal on graph, from Laplacian operator to Fourier transform

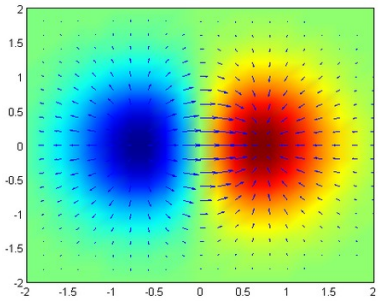


图 13: Gradient of the 2D function $f(x, y) = xe^{(x2+y2)}$ is plotted as blue arrows over the pseudocolor plot of the function.

Neural Network on Graphs XI

Spectral signal on graph, from Laplacian operator to Fourier transform

Consider a surface whose height above sea level at point (x, y) is $H(x, y)$. The gradient of H at a point is a vector pointing in the direction of the steepest slope or grade at that point. The steepness of the slope at that point is given by the magnitude of the gradient vector.

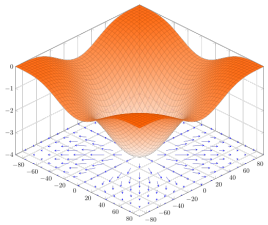


图 14: The gradient of the function $f(x, y) =?(\cos^2 x + \cos^2 y)^2$ depicted as a projected vector field on the bottom plane.

Neural Network on Graphs XII

Spectral signal on graph, from Laplacian operator to Fourier transform

In the three-dimensional Cartesian coordinate system with a Euclidean metric, the gradient, if it exists, is given by:

$$\nabla f = \frac{\partial f}{\partial x}\mathbf{i} + \frac{\partial f}{\partial y}\mathbf{j} + \frac{\partial f}{\partial z}\mathbf{k} = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{pmatrix}.$$

For example, the gradient of the function $f(x, y, z) = 2x + 3y^2 - \sin(z)$ is

$$\nabla f = 2\mathbf{i} + 6y\mathbf{j} - \cos(z)\mathbf{k} = \begin{pmatrix} 2 \\ 6y \\ -\cos(z) \end{pmatrix}.$$

And generally, for a $x \in \mathbb{R}^n$ and a multi-variant function $f(x)$,

$$\nabla f(x) = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n})^T.$$

Neural Network on Graphs XIII

Spectral signal on graph, from Laplacian operator to Fourier transform

Divergence of a vector field (矢量场/向量场) is the measure of "Outgoingness" of the field at a given point [18].

$$\nabla \cdot \boldsymbol{E} = \lim_{\Delta v \to 0} \frac{\oint_s \boldsymbol{E} ds}{\Delta \boldsymbol{v}} \tag{9}$$

Note: Divergence defined in equation (8) is a simple form of equation (9) on Cartesian space.

How to understanding divergence [19]?
For a basic understanding of divergence, it's enough to see that if a fluid is expanding (i.e., the flow has positive divergence everywhere inside the sphere), the net flow out of a sphere will be positive.

Neural Network on Graphs XIV

Spectral signal on graph, from Laplacian operator to Fourier transform

Take an example in $\mathbb{R}^2$, the divergence of a vector field is relatively easy to understand intuitively. Imagine that the vector field F pictured below gives the velocity of some fluid flow. It appears that the fluid is exploding outward from the origin.

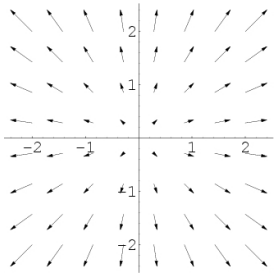


图 15: An example of vector field explosion.

Neural Network on Graphs XV

Spectral signal on graph, from Laplacian operator to Fourier transform

Based on the above illustration, one can see that Laplace Operator is a 2-order operator working on n-dimensional Euclidean space.
A simple definition of Δf or Δ in 3D space $\mathbb{R}^3$ is:

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2},$$

or

$$\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}.$$

Furthermore, its generative form in $\mathbb{R}^n$ is:

$$\Delta = \sum_i \frac{\partial^2}{\partial x_i^2},$$

where $f = f(x_1, x_2, \cdots, x_n)$ is a $\mathbb{R}^n \to \mathbb{R}$ multi variate function .

Neural Network on Graphs XVI

Spectral signal on graph, from Laplacian operator to Fourier transform

Discretion of differential operator and Laplacian operator:
For $f(x, y)$, one may approximate differential operator via:

$$\frac{\partial f}{\partial x} = f'_x(x, y) \approx f(x+1, y) - f(x, y),$$

and

$$\begin{aligned} \frac{\partial^2 f}{\partial x^2} &= f'(x) \\ &\approx f'(x, y) - f'(x-1, y) \\ &= f(x+1, y) + f(x-1, y) - 2f(x, y) \end{aligned}.$$

Similarly, one observe the Laplacian operator on f in a **discrete** form as:

Neural Network on Graphs XVII

Spectral signal on graph, from Laplacian operator to Fourier transform

$$\begin{aligned}\Delta f &= \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \\ &\approx f(x+1,y) + f(x-1,y) - 2f(x,y) + f(x,y+1) + f(x,y-1) - 2f(x,y) \\ &= f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)\end{aligned} \tag{10}$$

It is intuitive to explain Δf in the manner of divergence.

1. If $\Delta f = 0$, it is roughly say that the vector potential (势) of point $(x, y, f(x,y))$ equals to the sum of the other four $(x+1, y, f(x+1,y))$, $(x-1, yf(x-1,y)$, $(x, y+1, f(x,y+1)$ and $(x, y-1, f(x,y-1)$. So there is no potential difference in the local area of $f(x,y)$
2. If $\Delta f > 0$, it is roughly to say that the potential of the center point $(x, y, f(x,y))$ is lower than the neighborhood.
3. Similarly in the case of $\Delta f < 0$.

Neural Network on Graphs XVIII

Spectral signal on graph, from Laplacian operator to Fourier transform

Expansion of Laplacian operator to a graph —Laplacian matrix
Assume there is a graph G with N nodes, so the above function f is not two-dimensional but N:

$$f = (f_1, f_2, ..., f_n),$$

f_i is the feature value in the i-th node. For example, f in figure 12 equals to $(2,3,2,3,3,1)$.
Motivated by equation (10), one defines the Laplacian operator Δ to graph G by

$$\Delta f_i = \sum_{j \in NE_i} (f_i - f_j), \tag{11}$$

where NE_i is the index set for all nodes in the neighborhood of node v_i.
Since $A_{ij} = 0$ if node v_i is not adjacent with node v_j, one have

Neural Network on Graphs XIX

Spectral signal on graph, from Laplacian operator to Fourier transform

$$\Delta f_i = d_i f_i - \sum_{j=1}^{N} a_{ij} f_j, \tag{12}$$

where $d_i = \#\{NE_i\}$ is the degree of the i-th node v_i. It is straightforward to observe that:

$$\Delta f = diag(d_i) f - Af = (D - A) f = Lf \tag{13}$$

Thus we obtain the Laplacian operator on a graph **in the form of Laplacian matrix**.

Neural Network on Graphs XX

Spectral signal on graph, from Laplacian operator to Fourier transform

One can also obtain a modified version by assigning weight w_{ij} to all nodes v_j with $j \in NE_i$, with requirement of $\sum_j w_{ij} = d_i$.

$$\begin{aligned}\Delta f_i &= \sum_{j \in NE_i} w_{ij}(f_i - f_j) \\ &= \sum_{j=1}^{N} w_{ij} f_i - \sum_{j=1}^{N} w_{ij} f_j \\ &= d_i f_i - w_{i:}^T f\end{aligned} \tag{14}$$

It is straightforward to observe that:

$$\Delta f = diag(d_i) f - Wf = (D - W) f := \acute{L} f \tag{15}$$

Please note that

$$\acute{L} = D - W$$

Neural Network on Graphs XXI

Spectral signal on graph, from Laplacian operator to Fourier transform

is as well a modified definition of Laplacian matrix – **Combinatorial Graph Laplacian Matrix**[20], which as a conclusion satisfies

定理

$$\begin{aligned} \Delta f &= (D - W)f = \not{L} f, \\ \not{L} f_i &= \sum_{j \in NE_i} w_{ij}(f_i - f_j), \end{aligned} \tag{16}$$

and

$$f^T \not{L} f = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2, \tag{17}$$

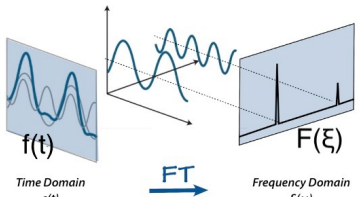
for a signal $f \in \mathbb{R}^n$ in graph.

Here equation (17) is a direct result from equation (2).

Neural Network on Graphs XXII

Spectral signal on graph, from Laplacian operator to Fourier transform

Fourier transform $\mathscr{F}$:



For a function $f : \mathbb{R} \longrightarrow \mathbb{C}$, the Fourier transform $\mathscr{F}$ convert $f(t)$ in a spatial domain to $F(\xi)$ in a spectral domain:

$$F(\xi) = \mathscr{F}[f(t)] = \int_{-\infty}^{+\infty} f(t) e^{-2\pi i \xi t} dt =< f(t), e^{-2\pi i \xi t} > . \tag{18}$$

Meanwhile, the inverse Fourier transform is:

$$f(t) = \mathscr{F}^{-1}[F(\xi)] = \int_{-\infty}^{+\infty} F(\xi) e^{2\pi i t \xi} d\xi =< F(\xi), e^{2\pi i t \xi} > . \tag{19}$$

Neural Network on Graphs XXIII

Spectral signal on graph, from Laplacian operator to Fourier transform

Since $e^{-2\pi i \xi t}$ is the eigen-function of the 1-dimensional Laplacian operator Δ, e.g.,

$$\Delta e^{-2\pi i \xi t} = \frac{\partial}{\partial t^2} e^{-2\pi i \xi t} = (2\pi\xi)^2 e^{-2\pi i \xi t},$$

analogously, one defines the Fourier transform $\mathscr{F}$ on graph as:

$$F(\lambda_l) = \mathscr{F}[f] =< f, u_l >= f^T u_l = \sum_{i=1}^{N} f_i u_{l;i}, \tag{20}$$

where $f = (f_1, f_2, \cdots, f_N)$ is the feature of all nodes, u_l is the eigen-vector of Laplacian operator Δ on graph, i.e.,

$$\Delta u_l = \not{L} u_l = \lambda u_l$$

.

Neural Network on Graphs XXIV

Spectral signal on graph, from Laplacian operator to Fourier transform

$$\underbrace{F(\lambda \cdot)}_{Analogy!} := \begin{pmatrix} F(\lambda_1) \\ F(\lambda_2) \\ \vdots \\ F(\lambda_N) \end{pmatrix} = \begin{pmatrix} u_{1;1} & u_{1;2} & \cdots & u_{1;N} \\ u_{2;1} & u_{2;2} & \cdots & u_{2;N} \\ \vdots & \vdots & \ddots & \vdots \\ u_{N;1} & u_{N;2} & \cdots & u_{N;N} \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{pmatrix} := U^T \underbrace{f(\cdot)}_{Analogy!}$$

Similarly, the inverse Fourier transform is:

$$\begin{aligned} & f_i = \mathscr{F}^{-1}[F(\lambda \cdot)] =< F(\lambda \cdot), u_l >= F(\lambda \cdot)^T u_l = \sum_{l=1}^{N} F(\lambda_l) u_{l;i}, \\ & \text{i.e.,} \\ & f(\cdot) = UF(\lambda .) \end{aligned} \tag{21}$$

Neural Network on Graphs XXV

Spectral signal on graph, from Laplacian operator to Fourier transform

Convolution $\star$ of two functions:

$$f(t) \star h(t) = \int_{-\infty}^{+\infty} f(\tau)h(t-\tau)d\tau = \int_{-\infty}^{+\infty} f(t-\tau)h(\tau)d\tau \tag{22}$$

定理 (Fourier transform on computing convolution of two functions.)

$$\begin{aligned} \mathscr{F}[f(t) \star h(t)] &= H(\xi)F(\xi) \\ f(t) \star h(t) &= \mathscr{F}^{-1}[\mathscr{F}(f(t))\mathscr{F}(g(t))] \end{aligned} \tag{23}$$

Neural Network on Graphs XXVI

Spectral signal on graph, from Laplacian operator to Fourier transform

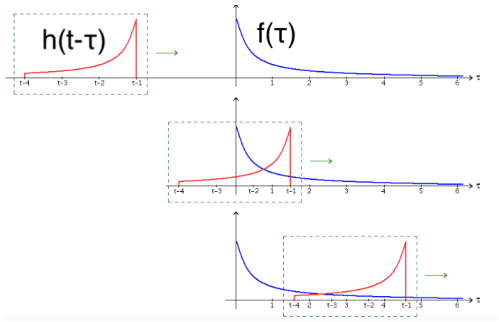


图 16: Visualization of convolution computation

As shown in the figure, $g(t)$ is actually a impulse signal, moving from $-\infty$ to $+\infty$ to detect $f(t)$.
Please notice that $< f, g >$ is keeping detected when g is moving.

Neural Network on Graphs XXVII

Spectral signal on graph, from Laplacian operator to Fourier transform

Motivated by the above definition, the convolution of two feature vector f and g in a graph $\mathcal{G}$ is defined as:

$$\begin{aligned} f(\cdot) \star h(\cdot) &= \mathscr{F}^{-1}[\mathscr{F}(h(\cdot)) \odot \mathscr{F}(f(\cdot))] = U(U^T h \odot U^T f) \\ &= U((H(\lambda_1), H(\lambda_2), \cdot, H(\lambda_n))^T \odot U^T f) \\ &= U \begin{pmatrix} H(\lambda_1) & & \\ & \vdots & \\ & & H(\lambda_n) \end{pmatrix} U^T f, \end{aligned} \tag{24}$$

where $H(\lambda_l) = \sum_{i=1}^{N} h_i u_{l;i}$

Neural Network on Graphs XXVIII

Spectral signal on graph, from Laplacian operator to Fourier transform

Finally, one found that: To find a proper convolutional kernel $h(\cdot)$ in a graph is actually to learn the matrix $\begin{pmatrix} H(\lambda_1) & & \\ & \vdots & \\ & & H(\lambda_n) \end{pmatrix}$ by neural networks.

[14] http://www.ipam.ucla.edu/abstract/?tid=14506&pcode=DLT2018
[15] https://en.wikipedia.org/wiki/Laplacian_matrix
[16] In the mathematical field of graph theory, the degree matrix is a diagonal matrix which contains information about the degree of each vertex.
[17] In graph theory and computer science, an adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph.
[18] https://www.therightgate.com/divergence-vector-field-formulas/
[19] https://mathinsight.org/divergence_idea
[20] Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., and Vandergheynst, P. . The emerging field of signal processing on graphs: extending high-dimensional data analysis to networks and other irregular domains. IEEE Signal Processing Magazine, 30(3), 83-98. https://arxiv.org/pdf/1211.0053.pdf

Neural Network on Graphs I

GCN and cluster-GCN

In a semi-supervised node classification case, the **General graph Laplacian regularization term in the loss function** is :

$$\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{reg}, \\ \text{with } \mathcal{L}_{reg} = \sum_{i,j} A_{ij} ||f(X_i) - f(X_j)||^2 = f(X)^T L f(X) \tag{25}$$

Here, $\mathcal{L}_0$ denotes the supervised loss w.r.t. the labeled part of the graph,
$f(\cdot)$ can be a neural network like differentiable function,
λ is a weighing factor,
and X is a matrix of node feature vectors X_i,
$L = D - A$ denotes the unnormalized graph Laplacian of an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with N nodes $v_i \in \mathcal{V}$, edges $(v_i, v_j) \in \mathcal{E}$, an adjacency matrix $A \in \mathcal{R}^{n \times n}$ (binary or weighted) and a degree matrix, $D_{ii} = \sum_j A_{ij}$.
The $\mathcal{L}_{reg}$ is referred to Theorem 2.

Neural Network on Graphs II

GCN and cluster-GCN

Graph Convolutional Network (GCN) [21] presents a scalable approach for semi-supervised learning on graph-structured data that is based on an efficient **variant of convolutional neural networks** which **operate directly on graphs**.

GCN motivates the choice of their convolutional architecture via a localized first-order approximation of **spectral graph convolutions**.

The model scales **linearly** in the number of graph edges and **learns hidden layer representations** that encode both local graph structure and features of nodes.

Neural Network on Graphs III

GCN and cluster-GCN

定理 (The layer-wise propagation rule of GCN)

$$H^{(l+1)} = \sigma(\underbrace{\tilde{D}^{\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}}\, H^{(l)} W^{(l)}), \tag{26}$$

1. *Here, $\tilde{A} = A + I_N$ is the adjacency matrix of the undirected graph $\mathcal{G}$ with added self-connections.*
2. *I_N is the identity matrix.*
3. *$\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ is the degree matrix.*
4. *$W^{(l)}$ is a layer-specific trainable weight matrix.*
5. *$\sigma(\cdot)$ denotes an activation function, such as the $ReLU(\cdot) = \max(0, \cdot)$.*
6. *$H^{(l)} \in \mathbb{R}^{N \times D}$ is the matrix of activations in the l-th layer.*
7. *$H^{(o)} = X$.*

Neural Network on Graphs IV

GCN and cluster-GCN

I strongly suggest you to recall equation (4) in standard GNN, and find the relevance.
Below is the reason why Equation (26) develops like this.
Cluster-GCN [22]

[21] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," ICLR 2017, 2017.https://arxiv.org/pdf/1609.02907.pdf

[22] Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks, https://arxiv.org/abs/1905.07953?context=cs.AI, https://github.com/benedekrozemberczki/ClusterGCN

Neural Network on Graphs

R-GCN, GCN on relation

Relational Graph Convolutional Networks (R-GCNs) [23] were applied to two standard knowledge base completion tasks:

1. Link prediction (recovery of missing facts, i.e. subject-predicate-object triples)
2. Entity classification (recovery of missing entity attributes).

It uses the DistMult factorization as the scoring function.

[23] Schlichtkrull, M. , Kipf, T. N. , Bloem, P. , Rianne van den Berg, and Welling, M. . (2018). Modeling Relational Data with Graph Convolutional Networks. The Semantic Web. Springer, Cham. https://arxiv.org/pdf/1703.06103.pdf

Acknowledgement

谢谢！