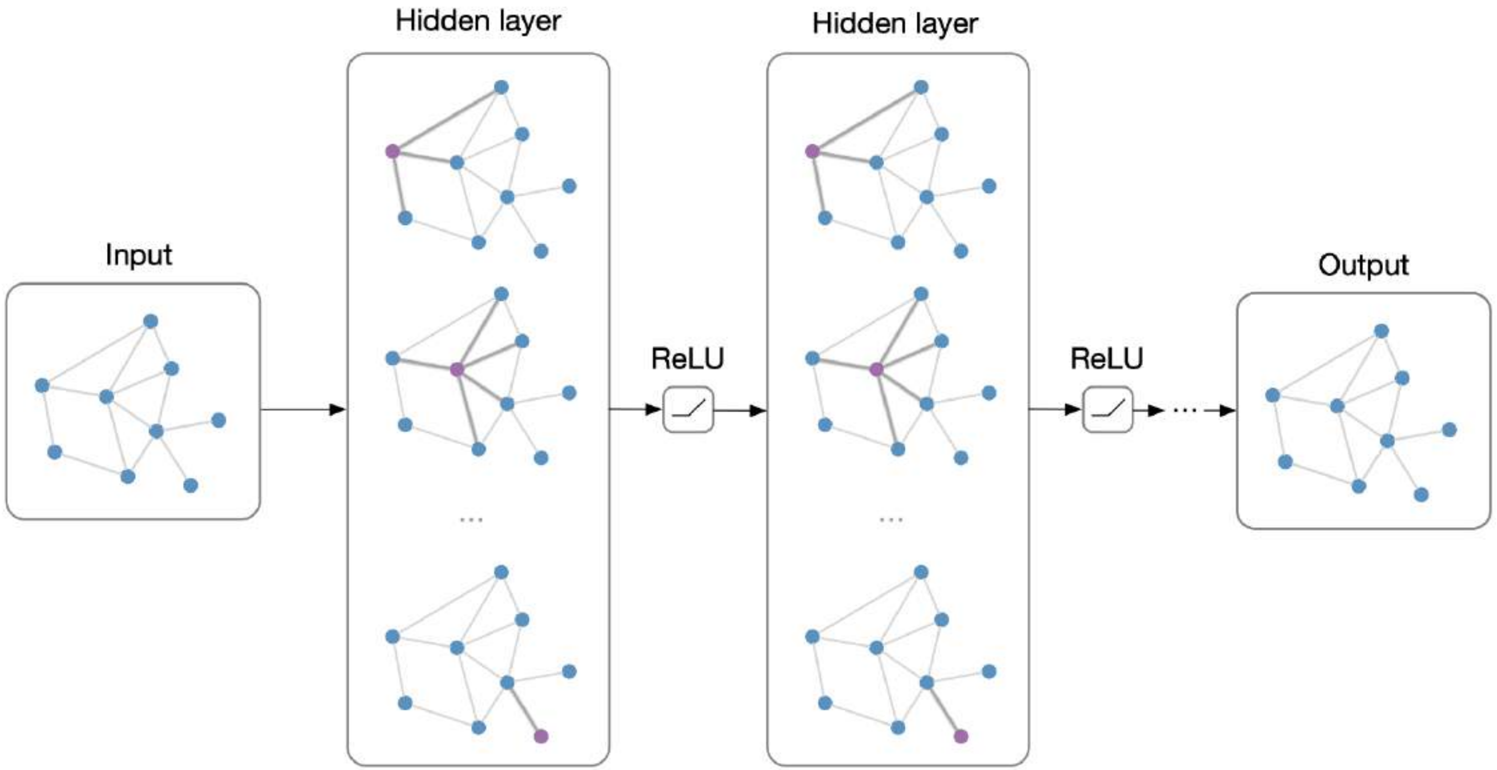
Graph Neural Networks



Xiachong Feng
TG
2019-04-08

Relies heavily on

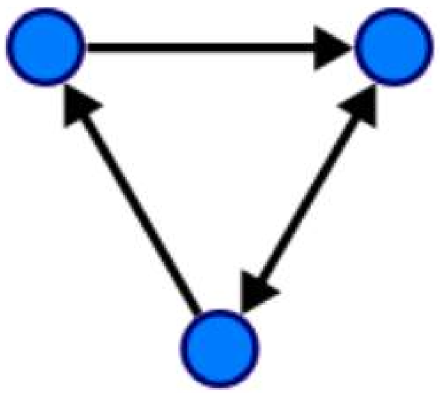
- A Gentle Introduction to Graph Neural Networks (Basics, DeepWalk, and GraphSage)
- Structured deep models: Deep learning on graphs and beyond
- Representation Learning on Networks
- Graph neural networks: Variations and applications
- http://snap.stanford.edu/proj/embeddings-www/
- Graph Neural Networks: A Review of Methods and Applications

Outline

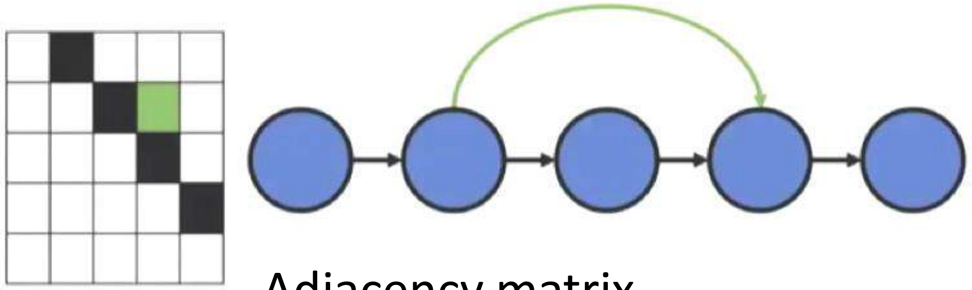
1. Basic && Overview
2. Graph Neural Networks
    1. Original Graph Neural Networks (GNNs)
    2. Graph Convolutional Networks (GCNs) && Graph SAGE
    3. Gated Graph Neural Networks (GGNNs)
    4. Graph Neural Networks With Attention (GAT)
    5. Sub-Graph Embeddings
3. Message Passing Neural Networks (MPNN)
4. GNN In NLP (AMR、SQL、Summarization)
5. Tools
6. Conclusion

Graph

- Graph is a data structure consisting of two components, vertices and edges.
- A graph G can be well described by the set of vertices V and edges E it contains.
- Edges can be either directed or undirected, depending on whether there exist directional dependencies between vertices.
- The vertices are often called nodes. these two terms are interchangeable.



$$G = (V, E)$$



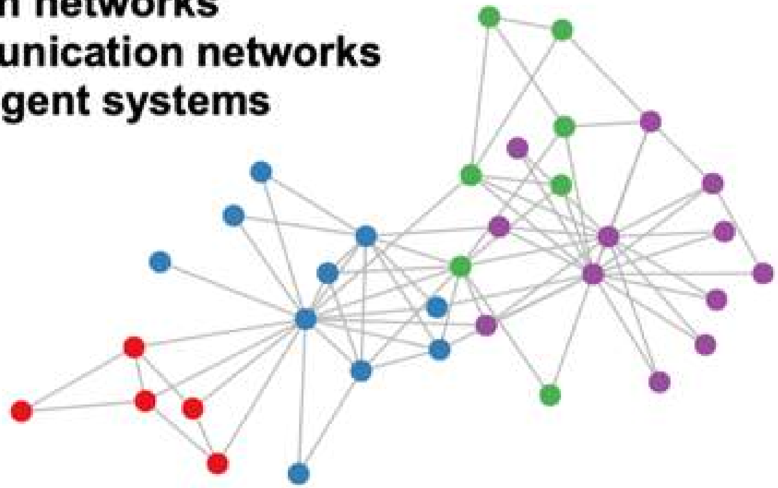
Adjacency matrix

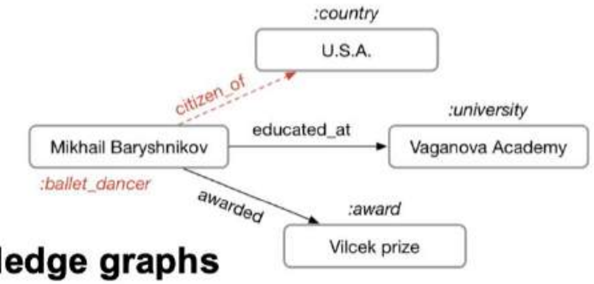
Graph structured data

A Gentle Introduction to Graph Neural Networks (Basics, DeepWalk, and GraphSage)

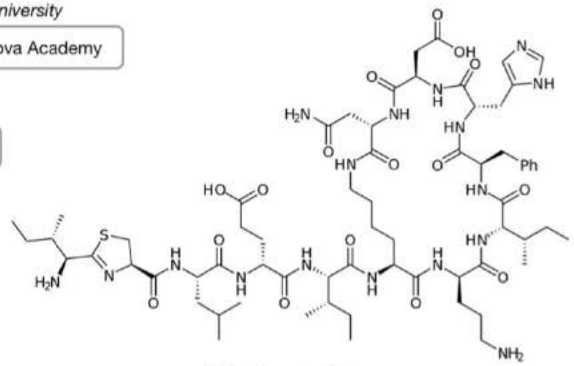
Graph-Structured Data

Social networks
Citation networks
Communication networks
Multi-agent systems

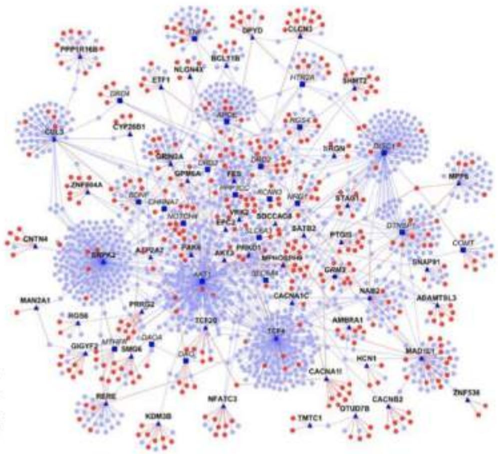




Knowledge graphs



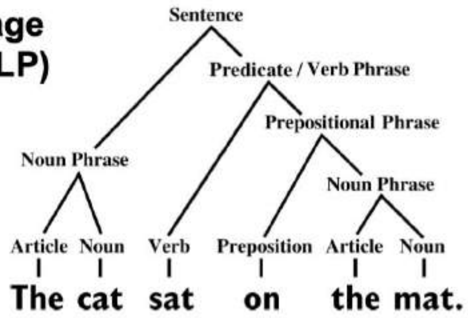
Molecules



Protein interaction networks

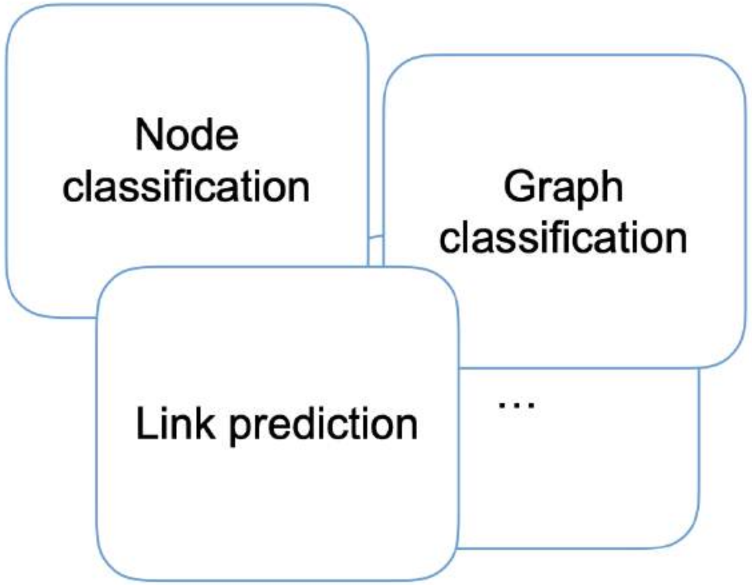
Road maps

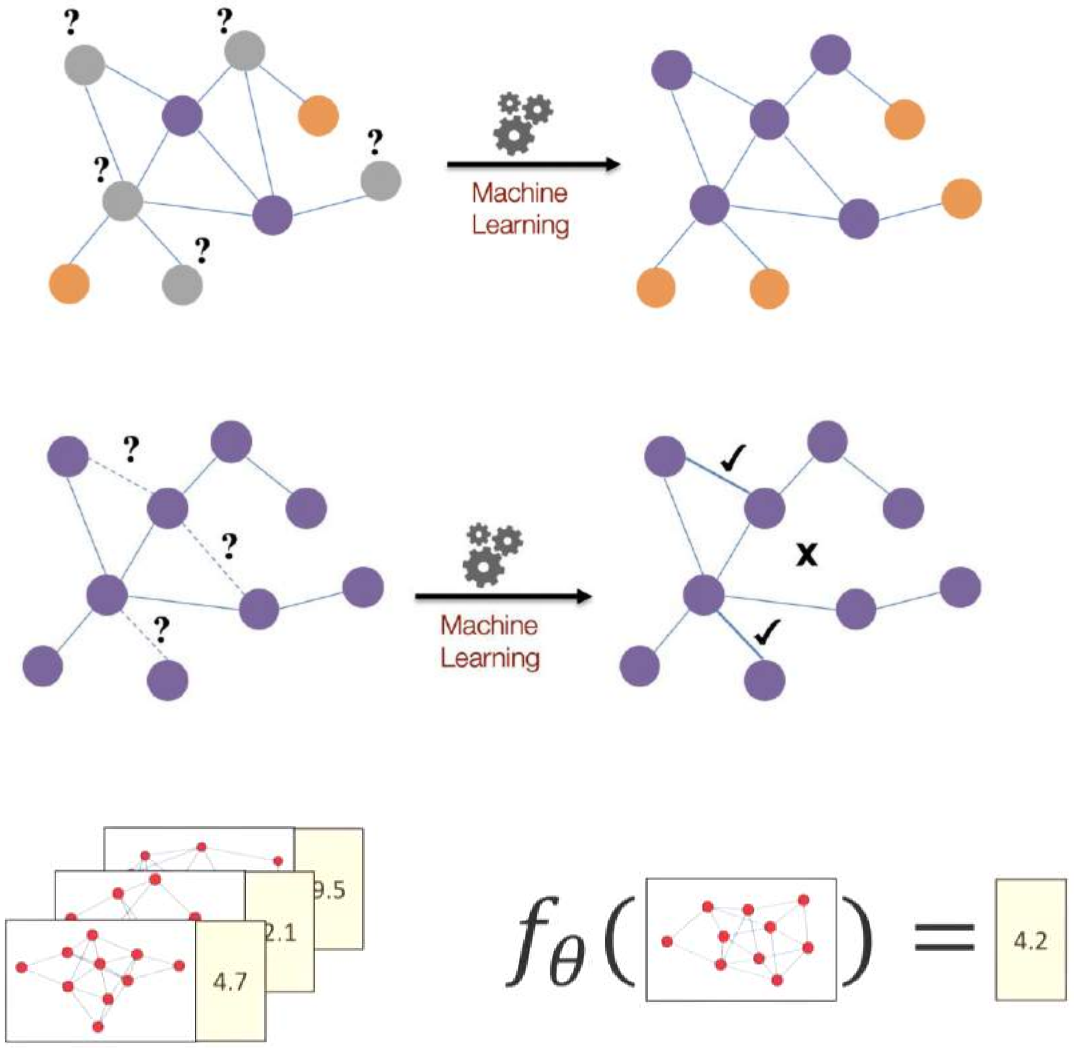
Natural language processing (NLP)



Structured deep models: Deep learning on graphs and beyond

Problems && Tasks

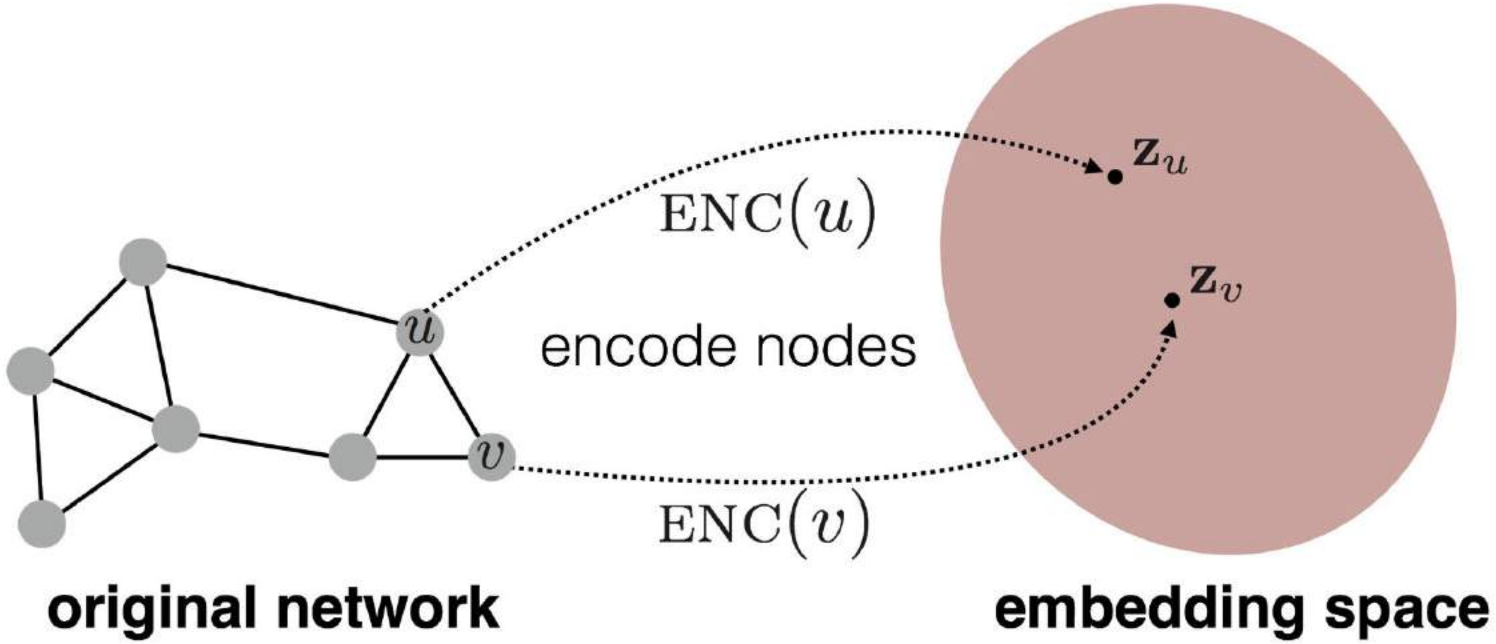




Representation Learning on Networks
Graph neural networks: Variations and applications

Embedding Nodes

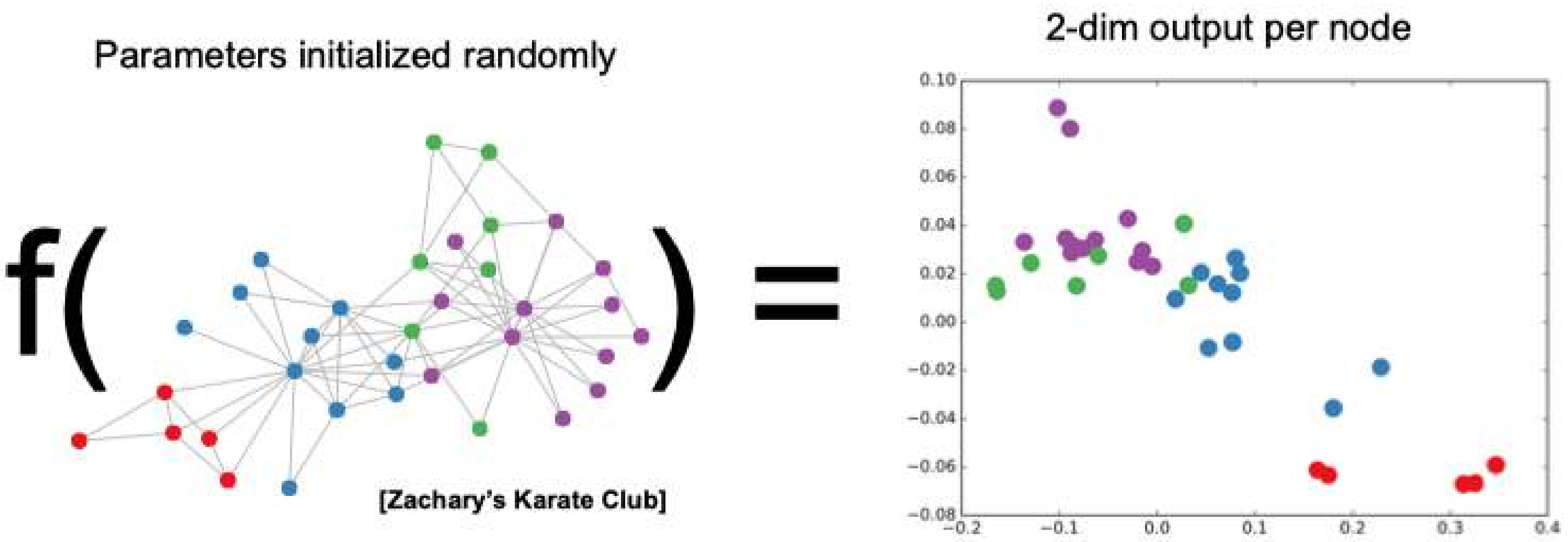
- Goal is to encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the original network.



http://snap.stanford.edu/proj/embeddings-www/

Embedding Nodes

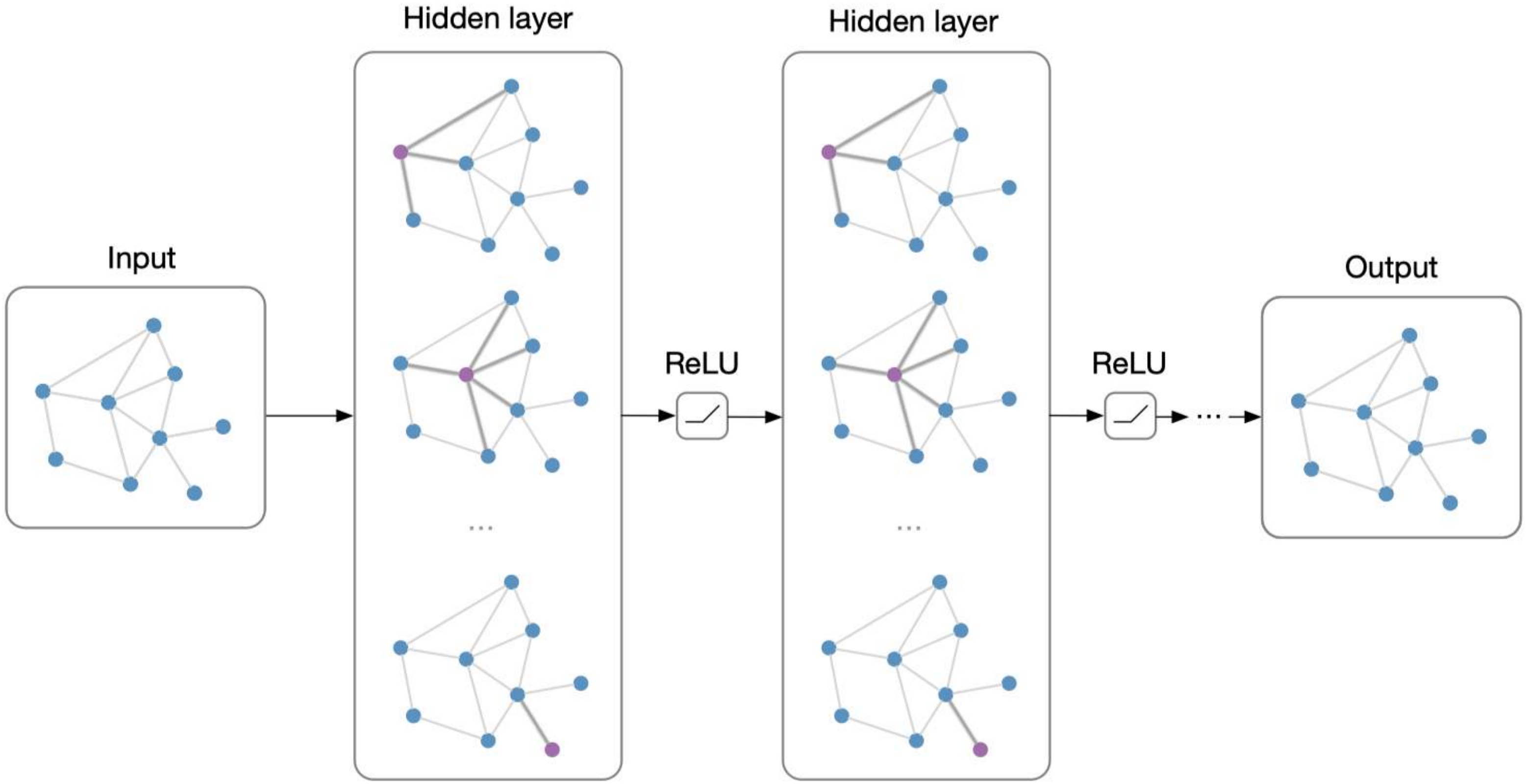
- Graph Neural Network is a neural network architecture that learns embeddings of nodes in a graph by looking at its nearby nodes.



http://snap.stanford.edu/proj/embeddings-www/

GNN Overview

The bigger picture:

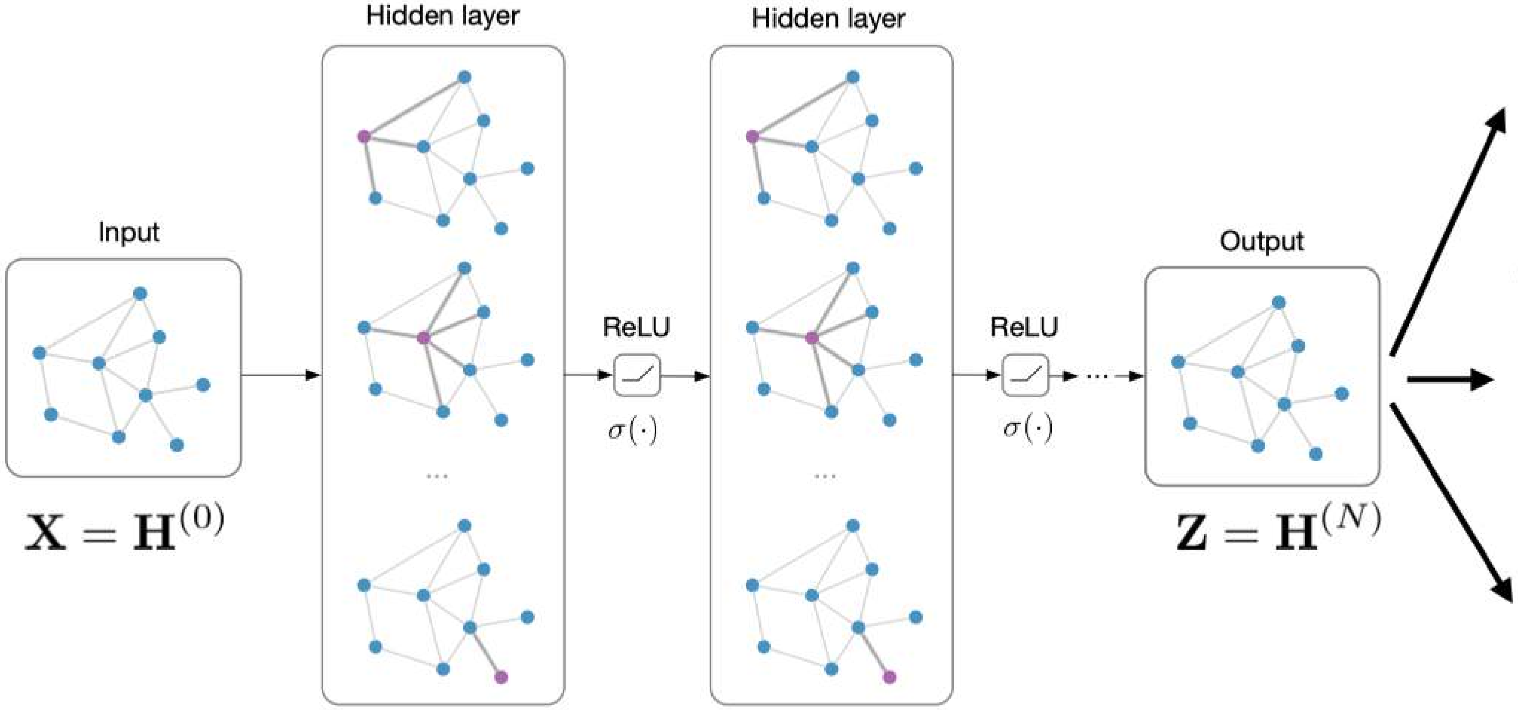


Main idea: Pass messages between pairs of nodes & agglomerate

Structured deep models: Deep learning on graphs and beyond

GNN Overview

Input: Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times E}$, preprocessed adjacency matrix $\hat{\mathbf{A}}$



Node classification:

$$\mathrm{softmax}(\mathbf{z_n})$$

e.g. Kipf & Welling (ICLR 2017)

Graph classification:

$$\mathrm{softmax}(\textstyle\sum_n \mathbf{z_n})$$

e.g. Duvenaud et al. (NIPS 2015)

Link prediction:

$$p(A_{ij}) = \sigma(\mathbf{z_i^T z_j})$$

Kipf & Welling (NIPS BDL 2016)
"Graph Auto-Encoders"

Structured deep models: Deep learning on graphs and beyond

Why GNN?

- Firstly, the standard neural networks like CNNs and RNNs cannot handle the graph input properly in that they stack the feature of nodes by a specific order. To solve this problem, GNNs propagate on each node respectively, ignoring the input order of nodes.
- Secondly, GNNs can do propagation guided by the graph structure, Generally, GNNs update the hidden state of nodes by a weighted sum of the states of their neighborhood.
- Thirdly, reasoning. GNNs explore to generate the graph from non-structural data like scene pictures and story documents, which can be a powerful neural model for further high-level AI.

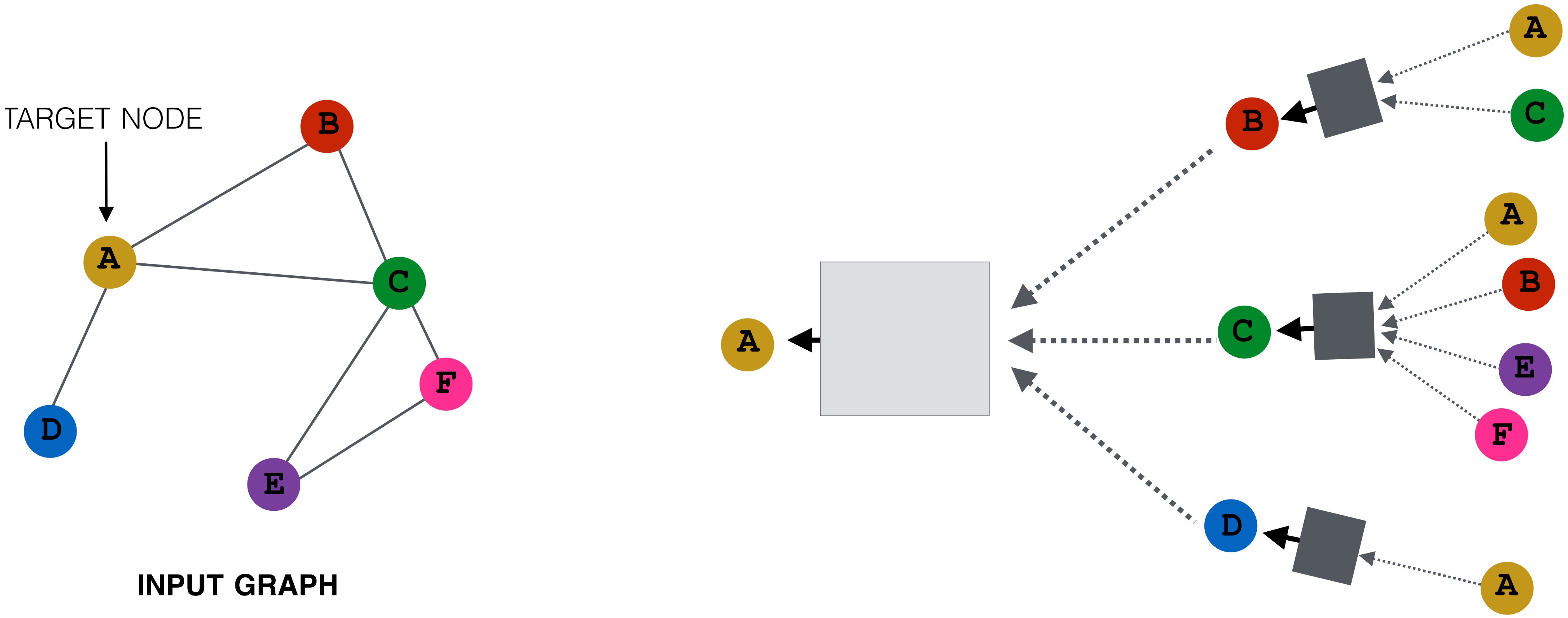
Graph Neural Networks: A Review of Methods and Applications

Outline

1. Basic && Overview
2. Graph Neural Networks
    1. Original Graph Neural Networks (GNNs)
    2. Graph Convolutional Networks (GCNs) && Graph SAGE
    3. Gated Graph Neural Networks (GGNNs)
    4. Graph Neural Networks With Attention (GAT)
    5. Sub-Graph Embeddings
3. Message Passing Neural Networks (MPNN)
4. GNN In NLP (AMR、SQL、Summarization)
5. Tools
6. Conclusion

Original Graph Neural Networks (GNNs)

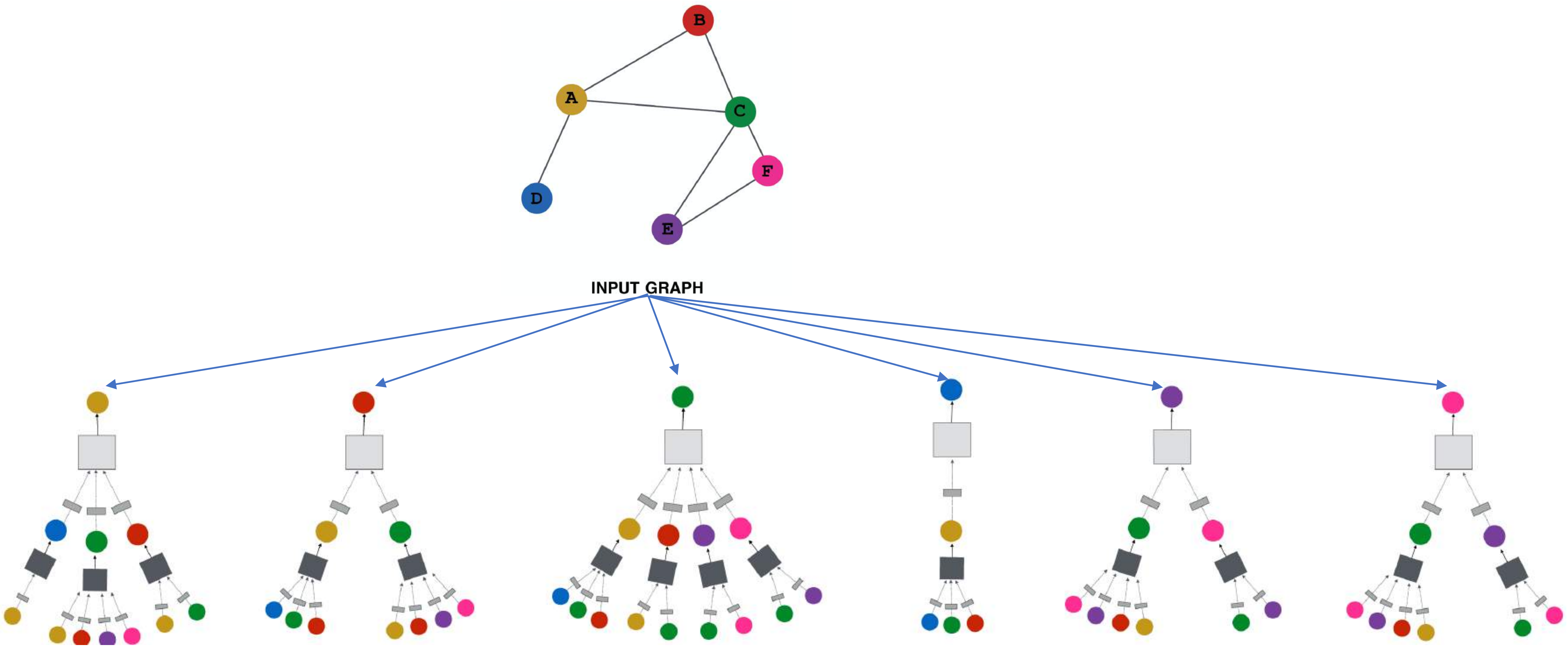
- Key idea: Generate node embeddings based on local neighborhoods.
- Intuition: Nodes aggregate information from their neighbors using neural networks



http://snap.stanford.edu/proj/embeddings-www/

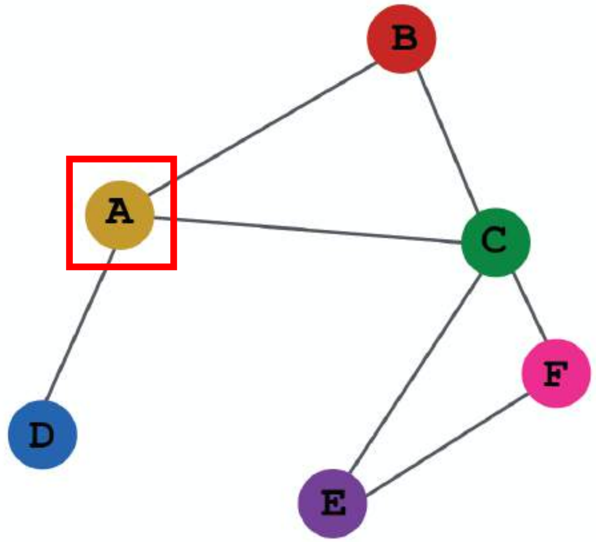
Original Graph Neural Networks (GNNs)

- Intuition: Network neighborhood defines a computation graph



http://snap.stanford.edu/proj/embeddings-www/

Original Graph Neural Networks (GNNs)



INPUT GRAPH

features of v   features of edges   neighborhood states   neighborhood features

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

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

local transition function   local output function

f and g can be interpreted as the feedforward neural networks.

$$\mathbf{H} = F(\mathbf{H}, \mathbf{X})$$

$$\mathbf{O} = G(\mathbf{H}, \mathbf{X}_N)$$

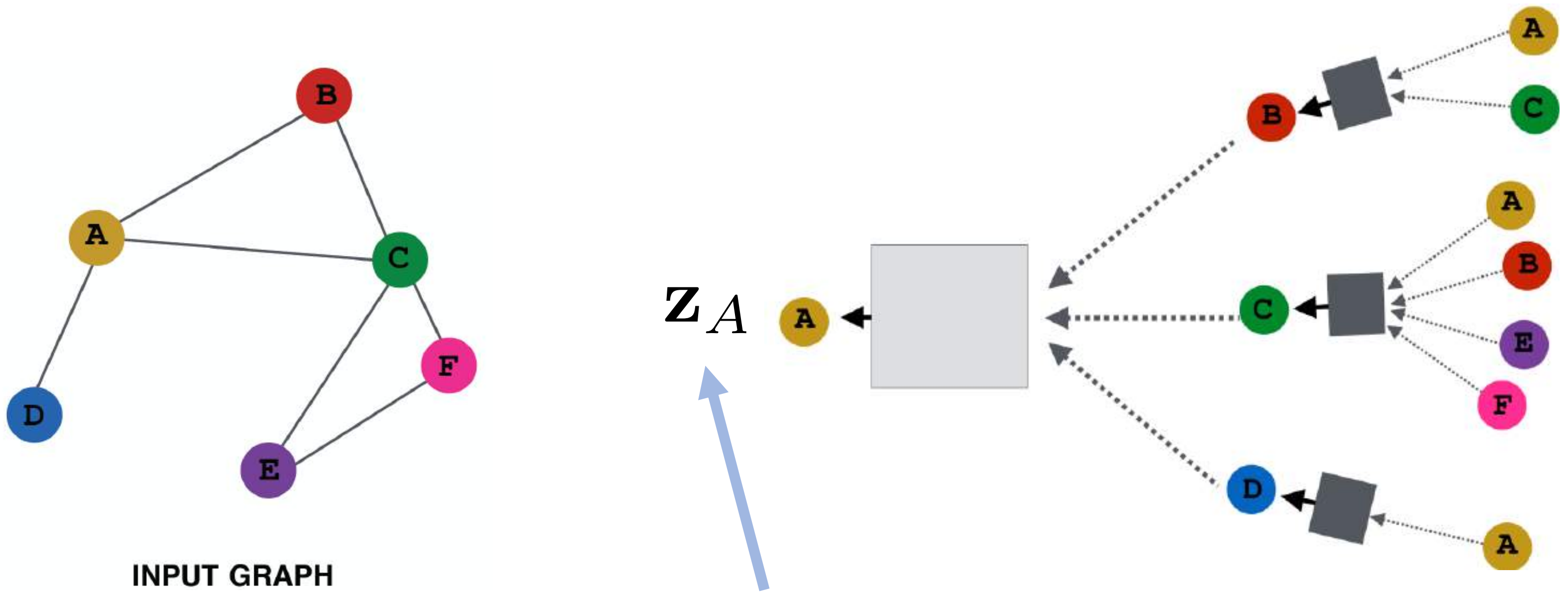
Banach`s fixed point theorem

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

http://snap.stanford.edu/proj/embeddings-www/

Original Graph Neural Networks (GNNs)

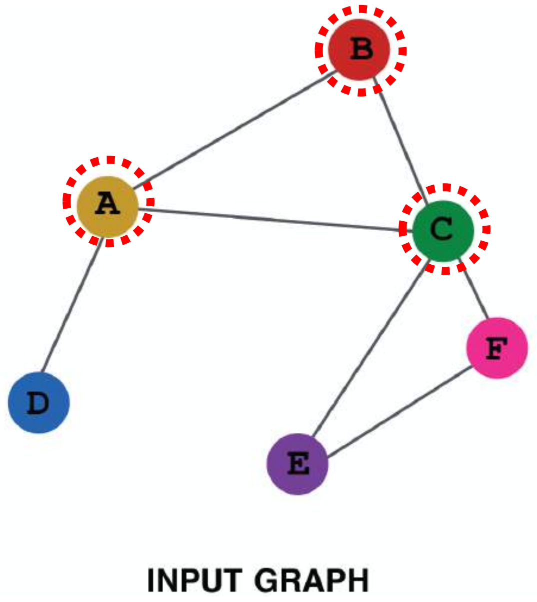
- How do we train the model to generate high-quality embeddings?



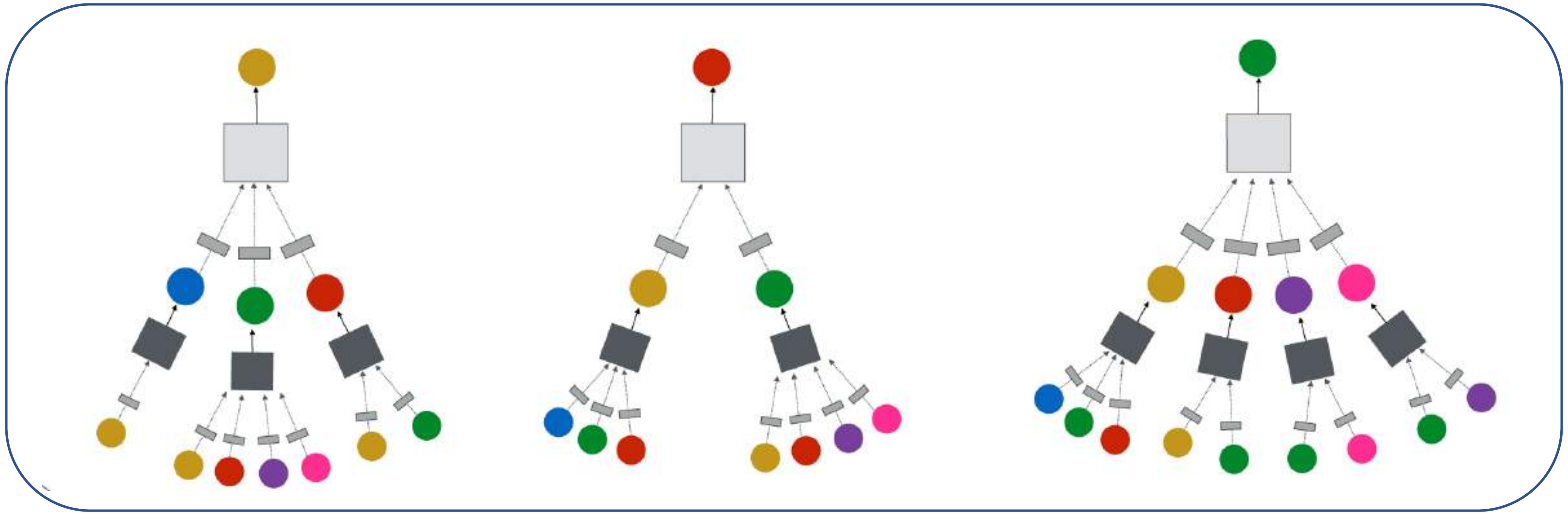
Need to define a loss function on the embeddings, L(z)!

Original Graph Neural Networks (GNNs)

- Train on a set of nodes, i.e., a batch of compute graphs



$$loss = \sum_{i=1}^{p} (\mathbf{t}_i - \mathbf{o}_i)$$



http://snap.stanford.edu/proj/embeddings-www/

Original Graph Neural Networks (GNNs)

Gradient-descent strategy

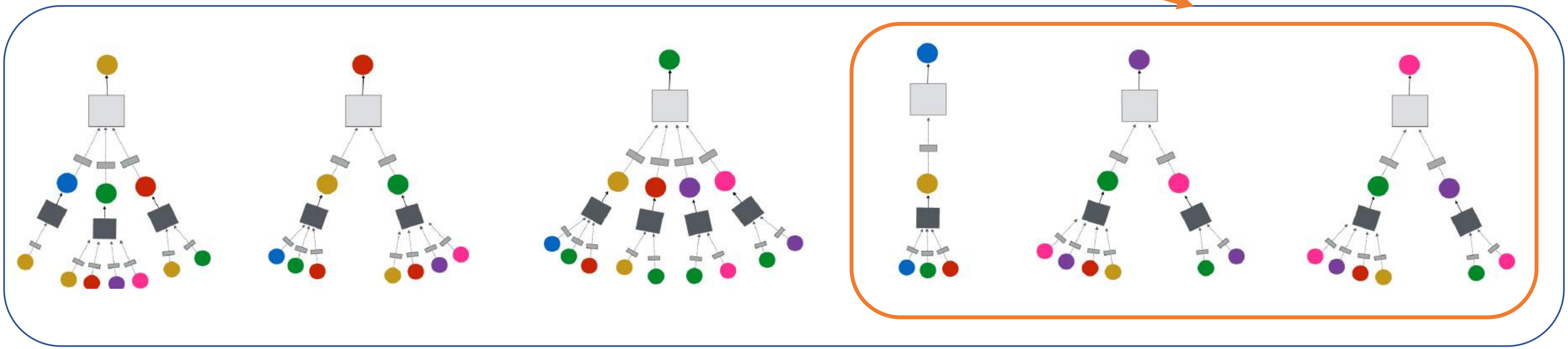
- The states h_v are iteratively updated by. $\mathbf{h}_v = f(\mathbf{x}_v, \mathbf{x}_{co[v]}, \mathbf{h}_{ne[v]}, \mathbf{x}_{ne[v]})$ a time T.

  They approach the fixed point solution of H(T) ≈ H.
- The gradient of weights W is computed from the loss.
- The weights W are updated according to the gradient computed in the last step.

Graph Neural Networks: A Review of Methods and Applications

Original Graph Neural Networks (GNNs)

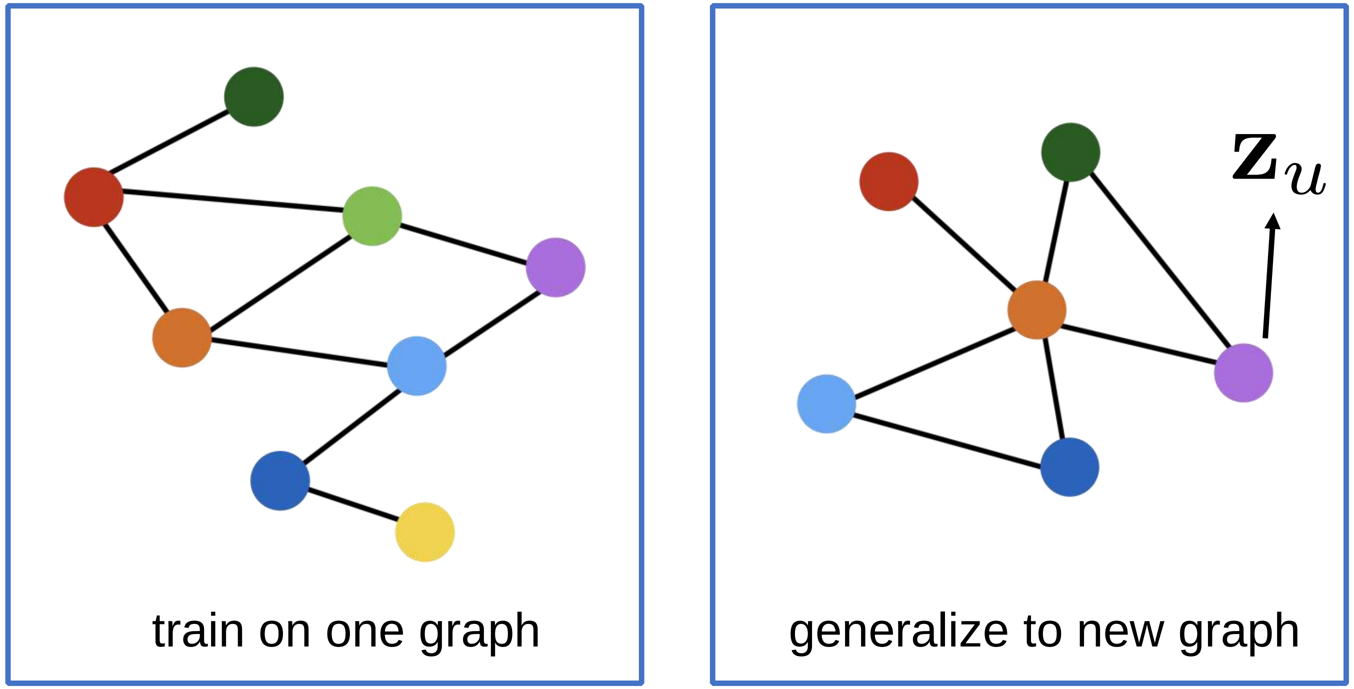
- Inductive Capability
  - Even for nodes we never trained on



http://snap.stanford.edu/proj/embeddings-www/

Original Graph Neural Networks (GNNs)

- Inductive Capability
  - Inductive node embedding-->generalize to entirely unseen graphs



http://snap.stanford.edu/proj/embeddings-www/

Original Graph Neural Networks (GNNs)

Limitations

- Firstly, it is inefficient to update the hidden states of nodes iteratively for the fixed point. If the assumption of fixed point is relaxed, it is possible to leverage Multi-layer Perceptron to learn a more stable representation, and removing the iterative update process. This is because, in the original proposal, different iterations use the same parameters of the transition function f, while the different parameters in different layers of MLP allow for hierarchical feature extraction.
- It cannot process edge information (e.g. different edges in a knowledge graph may indicate different relationship between nodes)
- Fixed point can discourage the diversification of node distribution, and thus may not be suitable for learning to represent nodes.

Average Neighbor Information

- Basic approach: Average neighbor information and apply a neural network.

Initial "layer 0" embeddings are equal to node features

$$\boxed{\mathbf{h}_v^0 = \mathbf{x}_v}$$

previous layer embedding of v

$$\boxed{\mathbf{h}_v^k} = \boxed{\sigma}\left(\mathbf{W}_k \boxed{\sum_{u\in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|}} + \mathbf{B}_k \boxed{\mathbf{h}_v^{k-1}}\right), \quad \forall k > 0$$

kth layer embedding of v

non-linearity (e.g., ReLU or tanh)

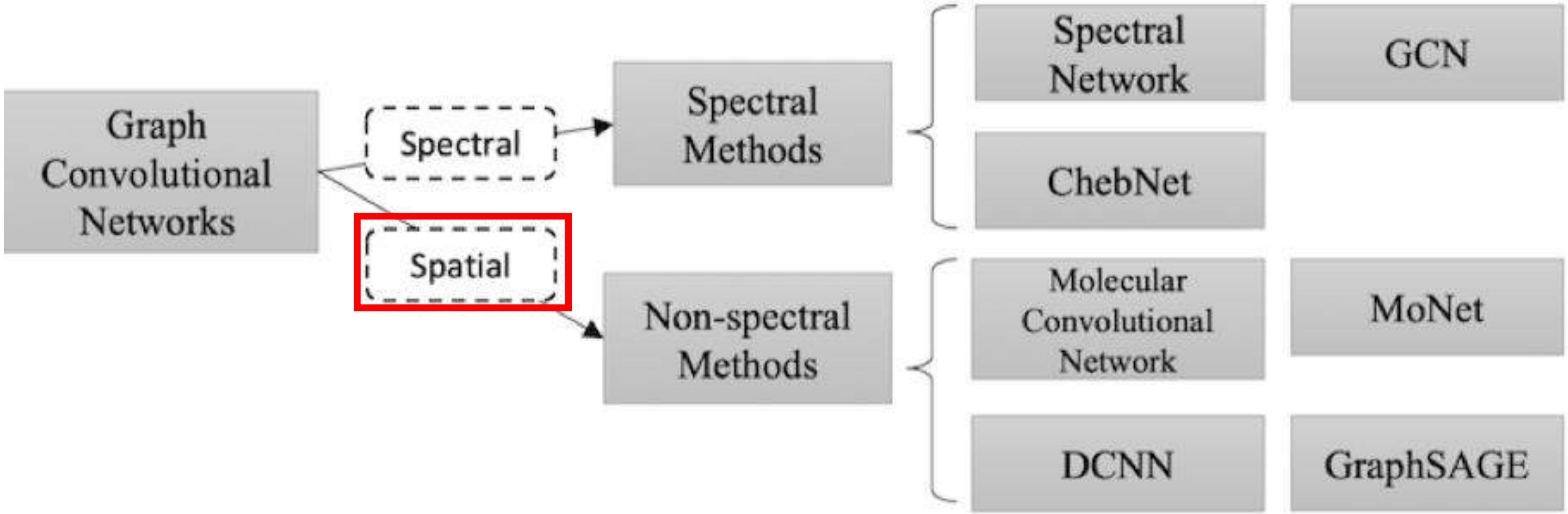
average of neighbor's previous layer embeddings

http://snap.stanford.edu/proj/embeddings-www/

Outline

1. Basic && Overview
2. Graph Neural Networks
    1. Original Graph Neural Networks (GNNs)
    2. Graph Convolutional Networks (GCNs) && Graph SAGE
    3. Gated Graph Neural Networks (GGNNs)
    4. Graph Neural Networks With Attention (GAT)
    5. Sub-Graph Embeddings
3. Message Passing Neural Networks (MPNN)
4. GNN In NLP (AMR、SQL、Summarization)
5. Tools
6. Conclusion

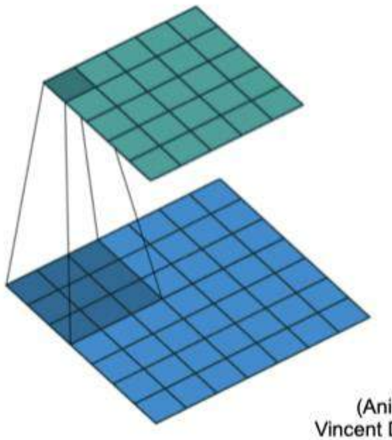
Graph Convolutional Networks (GCNs)



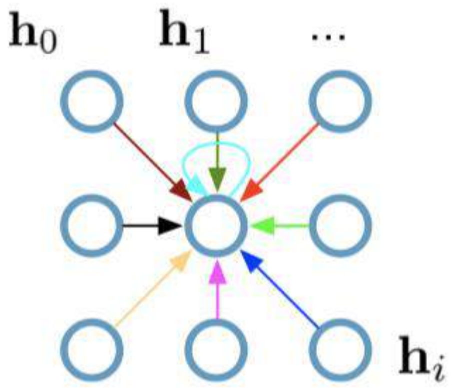
Graph Neural Networks: A Review of Methods and Applications

Convolutional Neural Networks (on grids)

Single CNN layer with 3x3 filter:



(Animation by Vincent Dumoulin)



Update for a single pixel:

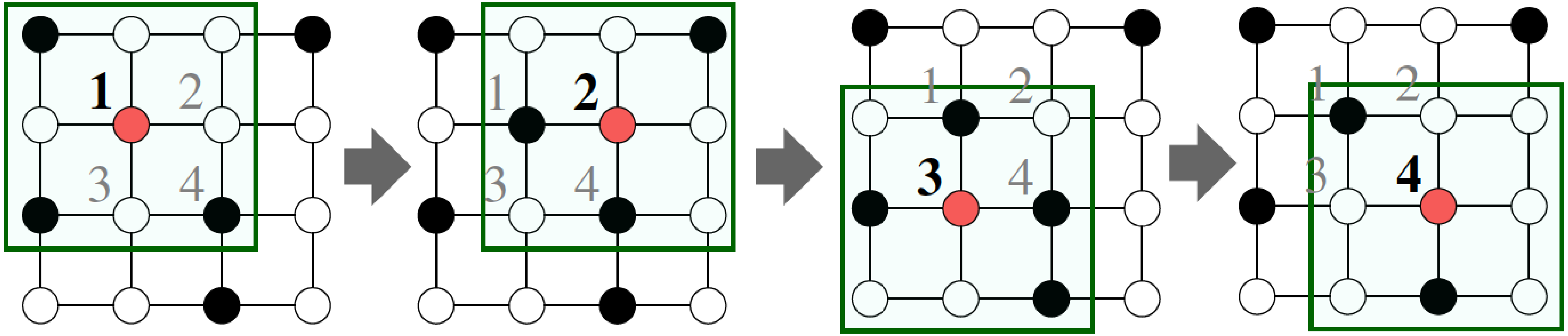
- Transform messages individually $\mathbf{W}_i \mathbf{h}_i$
- Add everything up $\sum_i \mathbf{W}_i \mathbf{h}_i$

$\mathbf{h}_i \in \mathbb{R}^F$ are (hidden layer) activations of a pixel/node

Full update:

$$\mathbf{h}_4^{(l+1)} = \sigma\left(\mathbf{W}_0^{(l)}\mathbf{h}_0^{(l)} + \mathbf{W}_1^{(l)}\mathbf{h}_1^{(l)} + \cdots + \mathbf{W}_8^{(l)}\mathbf{h}_8^{(l)}\right)$$

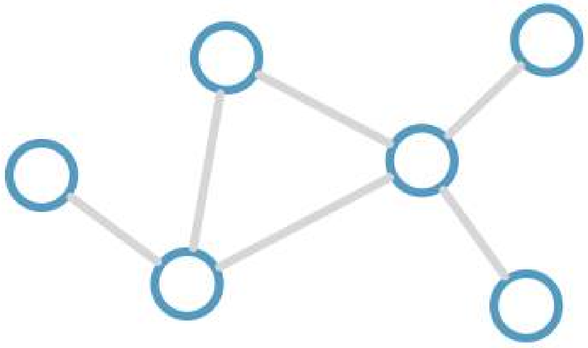
Convolutional Neural Networks (on grids)



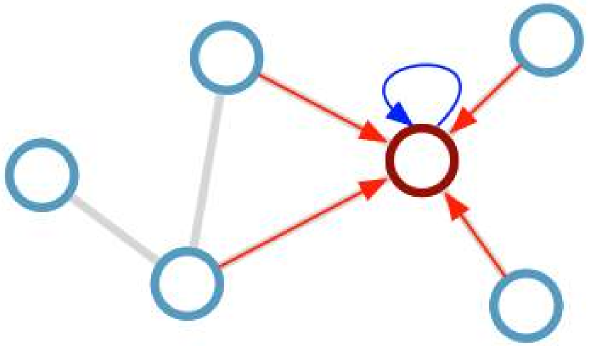
http://snap.stanford.edu/proj/embeddings-www/

Graph Convolutional Networks (GCNs)

Consider this undirected graph:



Calculate update for node in red:



Convolutional networks on graphs for learning molecular fingerprints *NIPS 2015*

$$\mathbf{x} = \mathbf{h}_v + \sum_{i=1}^{\mathcal{N}_v} \mathbf{h}_i$$

$$\mathbf{h}'_v = \sigma\left(\mathbf{x}\mathbf{W}_L^{\mathcal{N}_v}\right)$$

Update rule:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i} \frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

GraphSAGE

$$\mathbf{h}_{\mathcal{N}_v}^t = \text{AGGREGATE}_t\left(\{\mathbf{h}_u^{t-1}, \forall u \in \mathcal{N}_v\}\right)$$
$$\mathbf{h}_v^t = \sigma\left(\mathbf{W}^t \cdot [\mathbf{h}_v^{t-1} \| \mathbf{h}_{\mathcal{N}_v}^t]\right)$$

Mean aggregator.

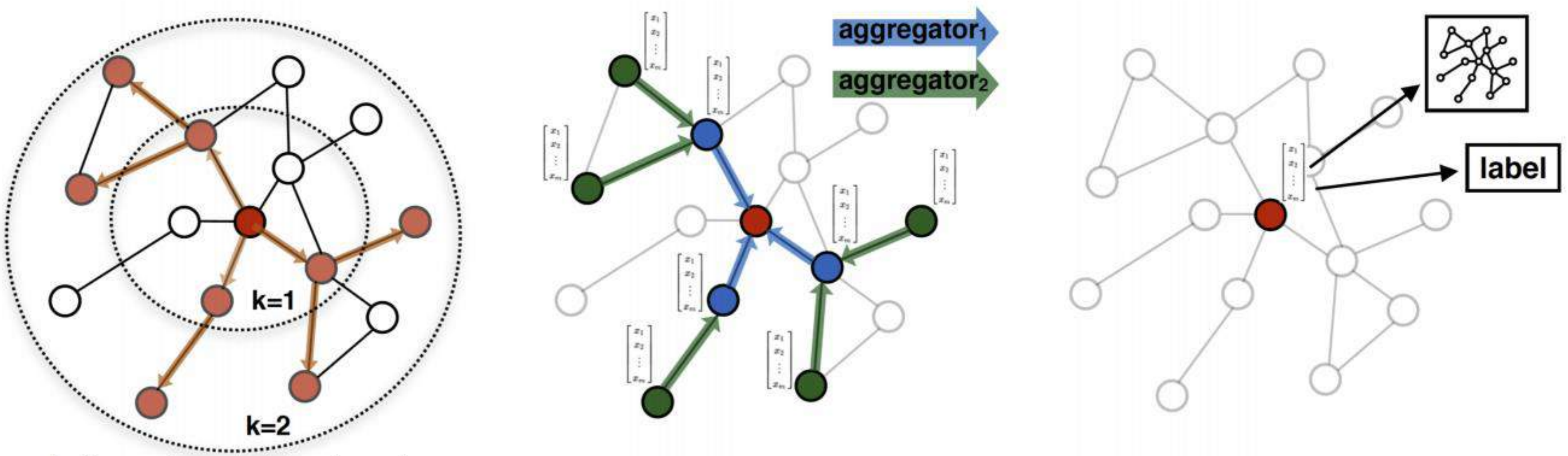
$$\mathbf{h}_v^t = \sigma(\mathbf{W} \cdot \text{MEAN}(\{\mathbf{h}_v^{t-1}\} \cup \{\mathbf{h}_u^{t-1}, \forall u \in \mathcal{N}_v\})$$

LSTM aggregator.

$$\text{AGG} = \text{LSTM}\left([\mathbf{h}_u^{k-1}, \forall u \in \pi(N(v))]\right)$$

Pooling aggregator.

$$\mathbf{h}_{\mathcal{N}_v}^t = \max(\{\sigma\left(\mathbf{W}_{\text{pool}}\mathbf{h}_u^{t-1} + \mathbf{b}\right), \forall u \in \mathcal{N}_v\})$$



1. Sample neighborhood

2. Aggregate feature information from neighbors

3. Predict graph context and label using aggregated information

Inductive Representation Learning on Large Graphs NIPS17

GraphSAGE

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

---

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$; *init*

for $k = 1...K$ **do** *K iters*

  for $v \in \mathcal{V}$ **do** *For every node*

    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$; *K-th func*

    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$

  end

  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$

end

$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

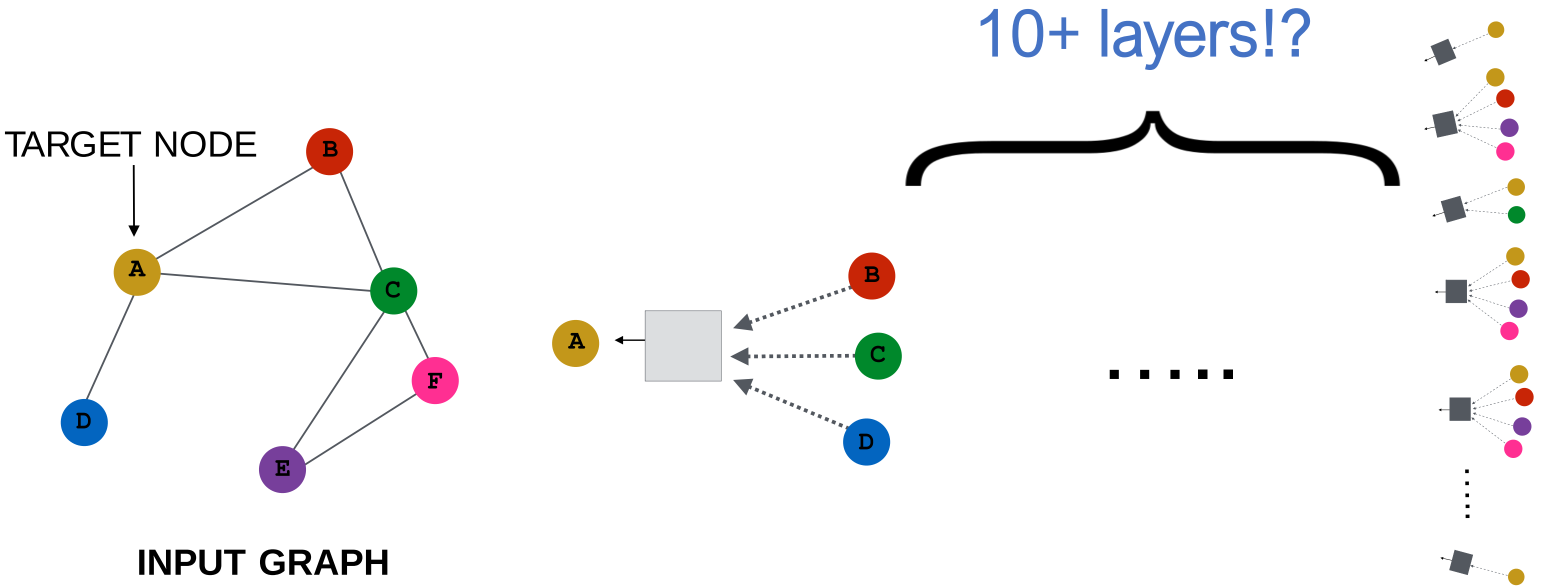
Inductive Representation Learning on Large Graphs NIPS17

Outline

1. Basic && Overview
2. Graph Neural Networks
    1. Original Graph Neural Networks (GNNs)
    2. Graph Convolutional Networks (GCNs) && Graph SAGE
    3. Gated Graph Neural Networks (GGNNs)
    4. Graph Neural Networks With Attention (GAT)
    5. Sub-Graph Embeddings
3. Message Passing Neural Networks (MPNN)
4. GNN In NLP (AMR、SQL、Summarization)
5. Tools
6. Conclusion

Gated Graph Neural Networks (GGNNs)

- GCNs and GraphSAGE generally only 2-3 layers deep.
- Challenges:
  - Overfitting from too many parameters.
  - Vanishing/exploding gradients during backpropagation.



http://snap.stanford.edu/proj/embeddings-www/

Gated Graph Neural Networks (GGNNs)

- GGNNs can be seen as **multi-layered GCNs** where **layer-wise parameters are tied and gating mechanisms** are added.

1. Get "message" from neighbors at step k:

$$\mathbf{m}_v^k = \boxed{\mathbf{W} \sum_{u \in N(v)} \mathbf{h}_u^{k-1}}$$

aggregation function does not depend on k

2. Update node "state" using Gated Recurrent Unit (GRU). New node state depends on the old state and the message from neighbors:

$$\mathbf{h}_v^k = \mathrm{GRU}(\mathbf{h}_v^{k-1}, \mathbf{m}_v^k)$$

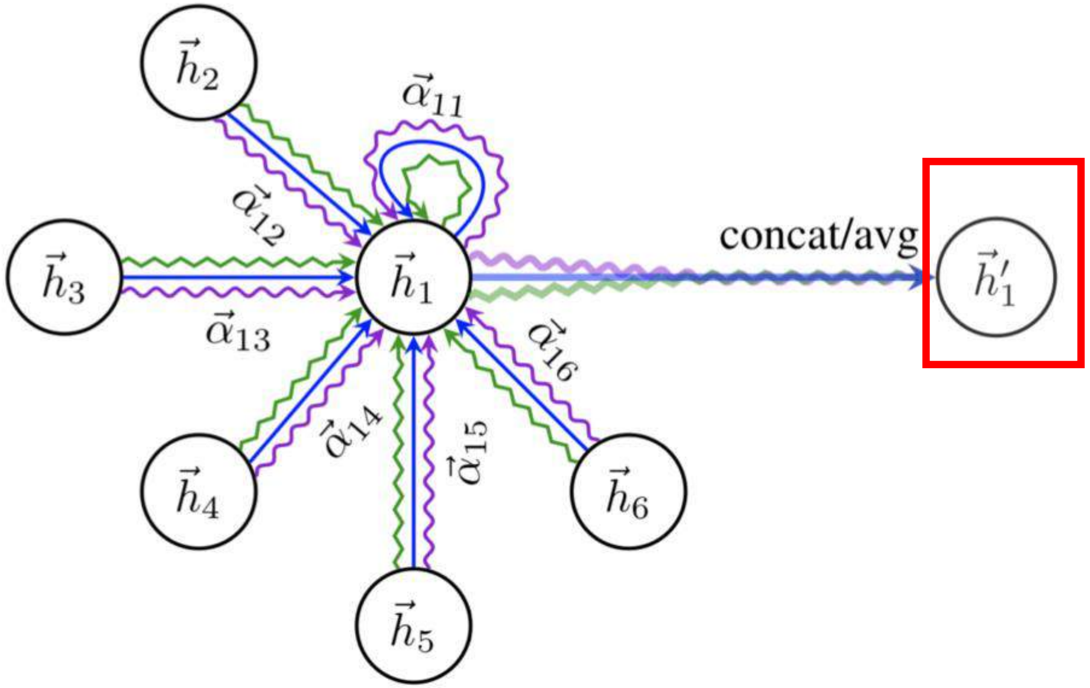
http://snap.stanford.edu/proj/embeddings-www/

Outline

1. Basic && Overview
2. Graph Neural Networks
   1. Original Graph Neural Networks (GNNs)
   2. Graph Convolutional Networks (GCNs) && Graph SAGE
   3. Gated Graph Neural Networks (GGNNs)
   4. Graph Neural Networks With Attention (GAT)
   5. Sub-Graph Embeddings
3. Message Passing Neural Networks (MPNN)
4. GNN In NLP (AMR、SQL、Summarization)
5. Tools
6. Conclusion

Graph Neural Networks With Attention

- Graph attention networks ICLR 2018 GAT



[Figure from Veličković et al. (ICLR 2018)]

$$\alpha_{ij} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\mathrm{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$

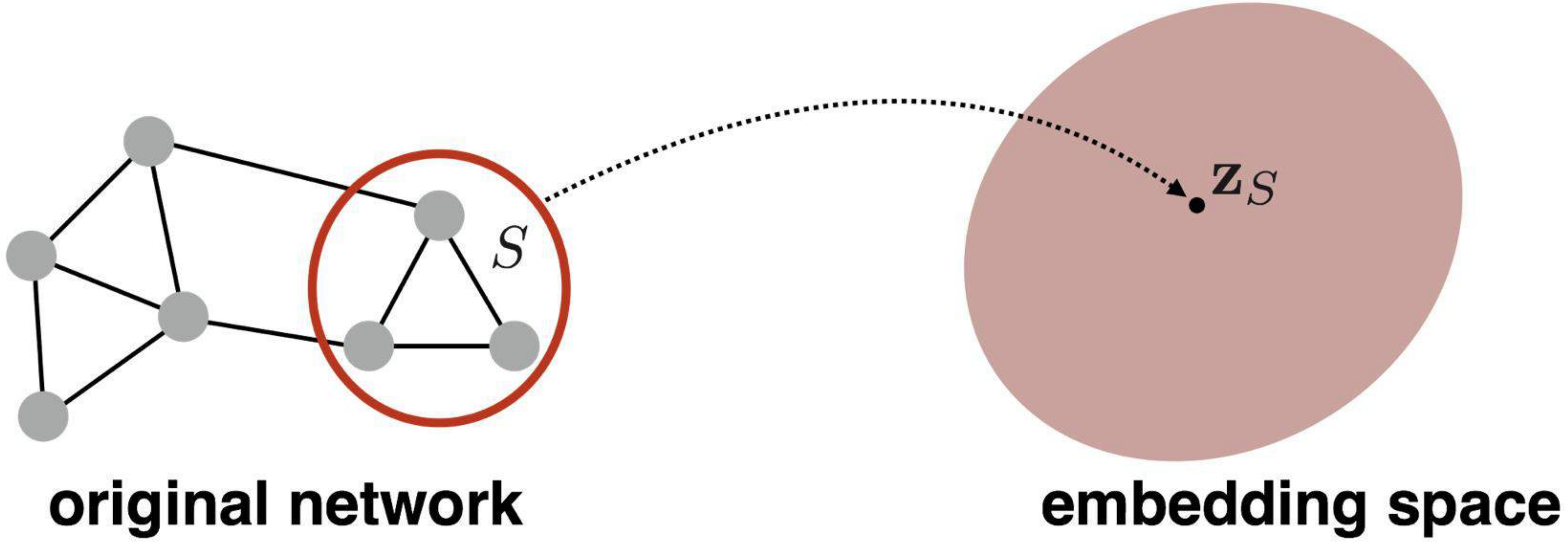
$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right)$$

Outline

1. Basic && Overview
2. Graph Neural Networks
    1. Original Graph Neural Networks (GNNs)
    2. Graph Convolutional Networks (GCNs) && Graph SAGE
    3. Gated Graph Neural Networks (GGNNs)
    4. Graph Neural Networks With Attention (GAT)
    5. Sub-Graph Embeddings
3. Message Passing Neural Networks (MPNN)
4. GNN In NLP (AMR、SQL、Summarization)
5. Tools
6. Conclusion

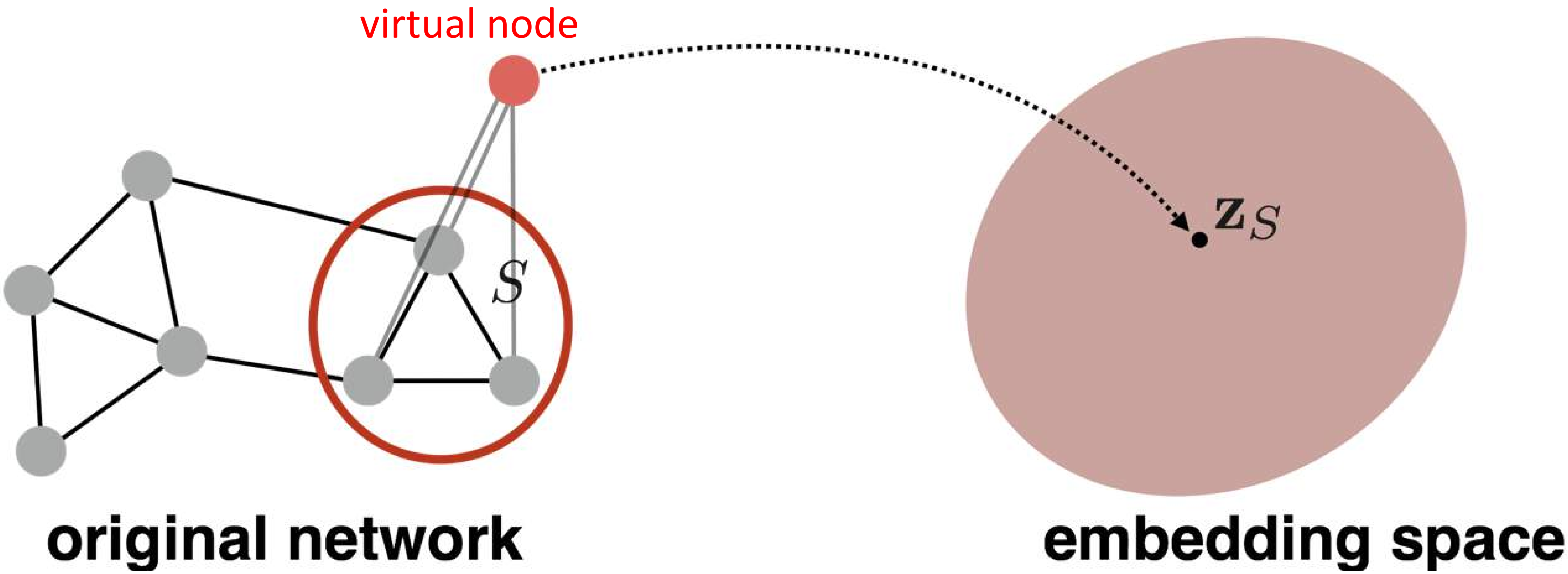
Sub-Graph Embeddings

$$\mathbf{z}_S = \sum_{v \in S} \mathbf{z}_v$$



http://snap.stanford.edu/proj/embeddings-www/

Sub-Graph Embeddings



http://snap.stanford.edu/proj/embeddings-www/

Outline

1. Basic && Overview
2. Graph Neural Networks
    1. Original Graph Neural Networks (GNNs)
    2. Graph Convolutional Networks (GCNs) && Graph SAGE
    3. Gated Graph Neural Networks (GGNNs)
    4. Graph Neural Networks With Attention (GAT)
    5. Sub-Graph Embeddings
3. Message Passing Neural Networks (MPNN)
4. GNN In NLP (AMR、SQL、Summarization)
5. Tools
6. Conclusion

Message Passing Neural Network (MPNN)

- Unified various graph neural network and graph convolutional network approaches.
- A general framework for supervised learning on graphs.
- Two phases, a message passing phase and a readout phase.
- Message passing phase (namely, the propagation step)
  - Runs for T time steps
  - Defined in terms of message function M_t and vertex update function U_t .
- Readout phase
  - computes a feature vector for the whole graph using the readout function R

$$\mathbf{m}_v^{t+1} = \sum_{w \in \mathcal{N}_v} M_t\left(\mathbf{h}_v^t, \mathbf{h}_w^t, \mathbf{e}_{vw}\right)$$

$$\mathbf{h}_v^{t+1} = U_t\left(\mathbf{h}_v^t, \mathbf{m}_v^{t+1}\right)$$

e_{vw} represents features of the edge from node v to w

$$\hat{\mathbf{y}} = R(\{\mathbf{h}_v^T | v \in G\})$$

Graph Neural Networks: A Review of Methods and Applications

MPNN && GGNN

$$M_t\left(\mathbf{h}_v^t, \mathbf{h}_w^t, \mathbf{e}_{vw}\right) = \mathbf{A}_{\mathbf{e}_{vw}} \mathbf{h}_w^t$$

$$U_t = GRU\left(\mathbf{h}_v^t, \mathbf{m}_v^{t+1}\right)$$

$$R = \sum_{v \in V} \sigma\left(i(\mathbf{h}_v^T, \mathbf{h}_v^0)\right) \odot \left(j(\mathbf{h}_v^T)\right)$$

Graph Neural Networks: A Review of Methods and Applications

Outline

1. Basic && Overview
2. Graph Neural Networks
    1. Original Graph Neural Networks (GNNs)
    2. Graph Convolutional Networks (GCNs) && Graph SAGE
    3. Gated Graph Neural Networks (GGNNs)
    4. Graph Neural Networks With Attention (GAT)
    5. Sub-Graph Embeddings
3. Message Passing Neural Networks (MPNN)
4. GNN In NLP (AMR、SQL、Summarization)
5. Tools
6. Conclusion

GNN IN NLP

- **AMR-To-Text**
  - A Graph-to-Sequence Model for AMR-to-Text Generation ***ACL 18***
  - Graph-to-Sequence Learning using Gated Graph Neural Networks ***ACL 18***
  - Structural Neural Encoders for AMR-to-text Generation ***NAACL 19***
- **SQL-To-Text**
  - SQL-to-Text Generation with Graph-to-Sequence Model ***EMNLP18***
- **Document Summarization**
  - Structured Neural Summarization ***ICLR 19***
  - Graph-based Neural Multi-Document Summarization ***CoNLL 17***

AMR

- Abstract Meaning Representation (AMR)
- **Graph**： rooted, directed graph
- **nodes** in the graph represent concepts and **edges** represent semantic relations between them
- **Task**： recover a text representing the same meaning as an input AMR graph.
- **Challenge**
  - word tenses and function words are abstracted away
- **Previous**
  - Seq2Seq Model
  - linearized AMR structure
  - **Problem**： closely-related nodes, such as parents, children and siblings can be far away after serialization.

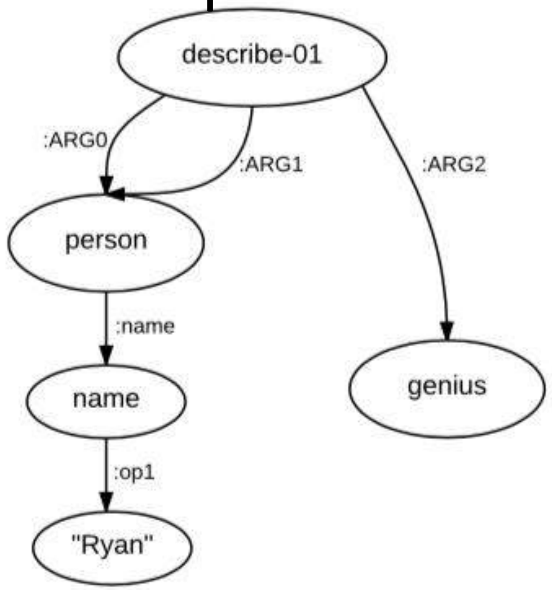


Figure 1: An example of AMR graph meaning "Ryan's description of himself: a genius."

AMR-to-Text

- Graph Encoder

$$G = (V, E) \quad g = \{h^j\}|_{v_j \in V}$$

Edge

$$x_j^i = \sum_{(i,j,l)\in E_{in}(j)} x_{i,j}^l$$

$$x_j^o = \sum_{(j,k,l)\in E_{out}(j)} x_{j,k}^l,$$

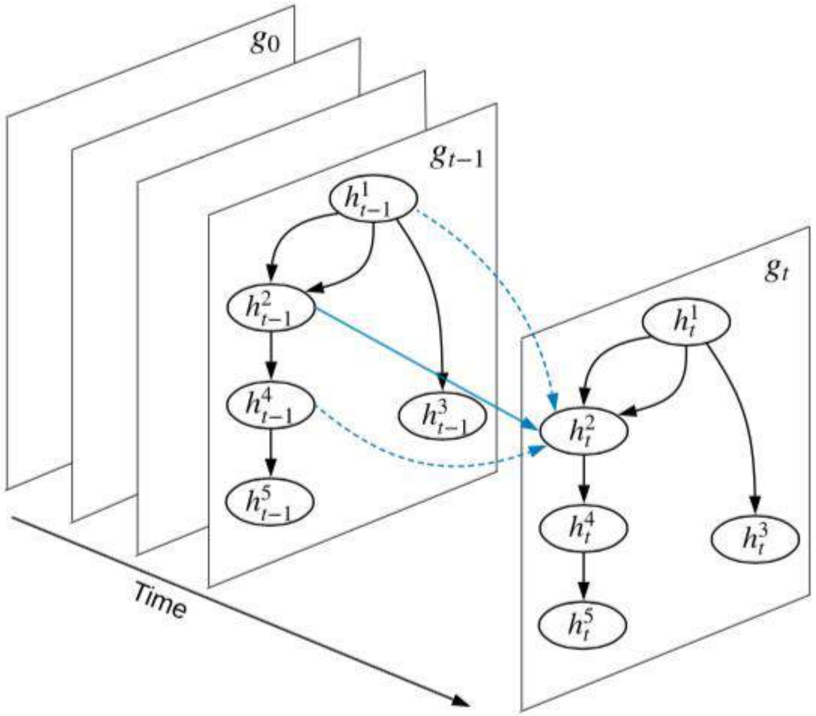
$$x_{i,j}^l = W_4\Big([e_l; e_i]\Big) + b_4$$

$$x_{i,j}^l = W_4\Big([e_l; e_i; h_i^c]\Big) + b_4,$$

Node

$$h_j^i = \sum_{(i,j,l)\in E_{in}(j)} h_{t-1}^i$$

$$h_j^o = \sum_{(j,k,l)\in E_{out}(j)} h_{t-1}^k,$$



$$g_t = \{h_t^j\}|_{v_j \in V}.$$

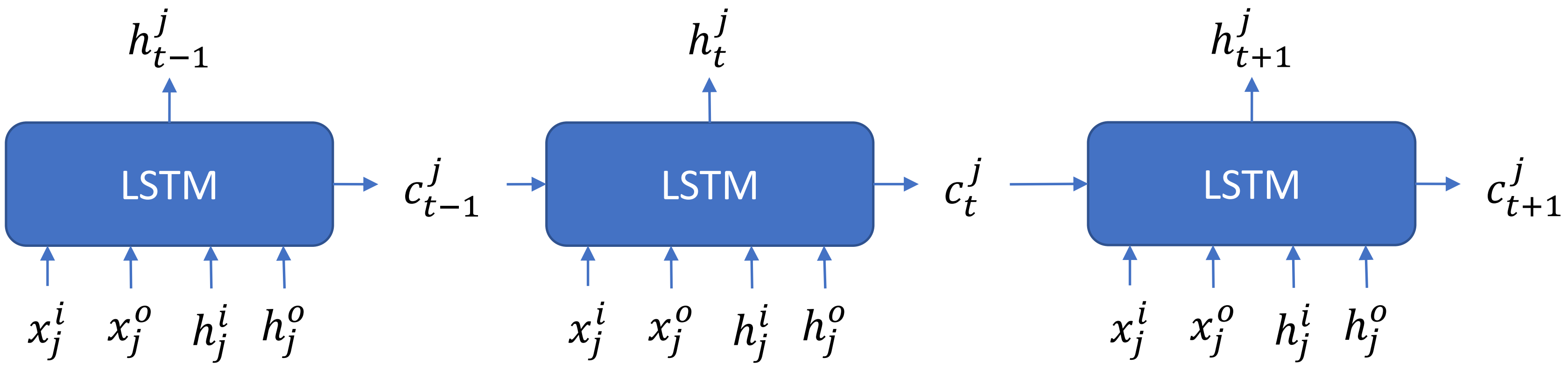
Graph Decoder

- Decoder initial state:average of the last states of all nodes.
- Each attention vector becomes $[h_T^j; x_j]$

A Graph-to-Sequence Model for AMR-to-Text Generation ACL 18

Can not learn Edge representations!

AMR-To-Text

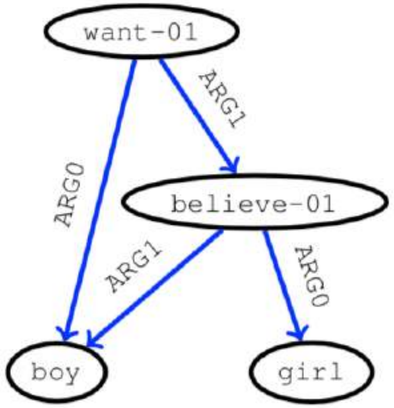


$$
\begin{aligned}
i^j_t &= \sigma(\boxed{W_i} x^i_j + \boxed{\hat{W}_i} x^o_j + \boxed{U_i} h^i_j + \boxed{\hat{U}_i} h^o_j + b_i),\\
o^j_t &= \sigma(\boxed{W_o} x^i_j + \boxed{\hat{W}_o} x^o_j + \boxed{U_o} h^i_j + \boxed{\hat{U}_o} h^o_j + b_o),\\
f^j_t &= \sigma(\boxed{W_f} x^i_j + \boxed{\hat{W}_f} x^o_j + \boxed{U_f} h^i_j + \boxed{\hat{U}_f} h^o_j + b_f),\\
u^j_t &= \sigma(\boxed{W_u} x^i_j + \boxed{\hat{W}_u} x^o_j + \boxed{U_u} h^i_j + \boxed{\hat{U}_u} h^o_j + b_u),\\
c^j_t &= f^j_t \odot c^j_{t-1} + i^j_t \odot u^j_t,\\
h^j_t &= o^j_t \odot \tanh(c^j_t),
\end{aligned}
$$

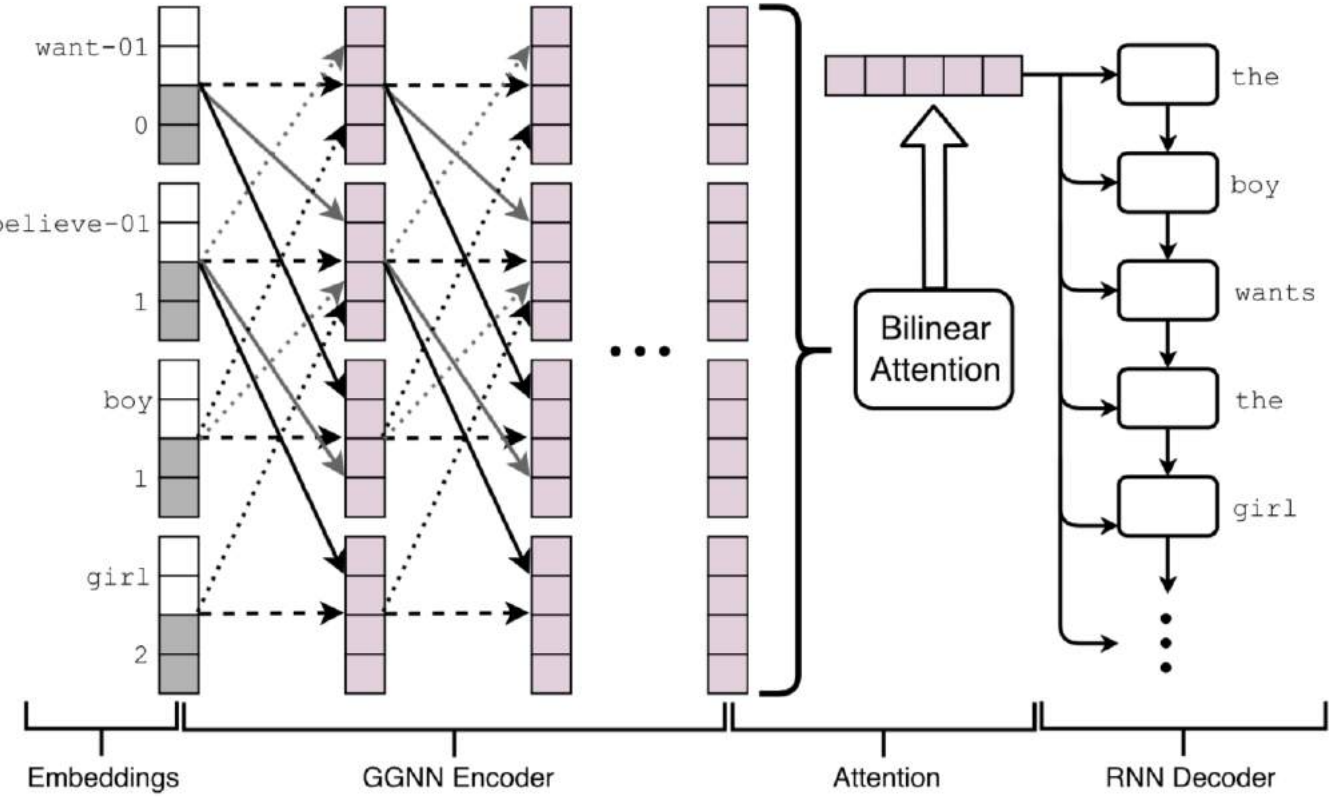
A Graph-to-Sequence Model for AMR-to-Text Generation ACL 18

AMR-to-Text

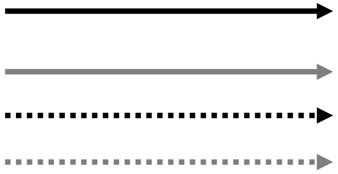
- Previous： represent edge information as label-wise parameters
- Nodes and edges to have their own hidden representations.
- Method： graph transformation that changes edges to additional nodes



The boy wants the girl to believe him



edge-wise parameters

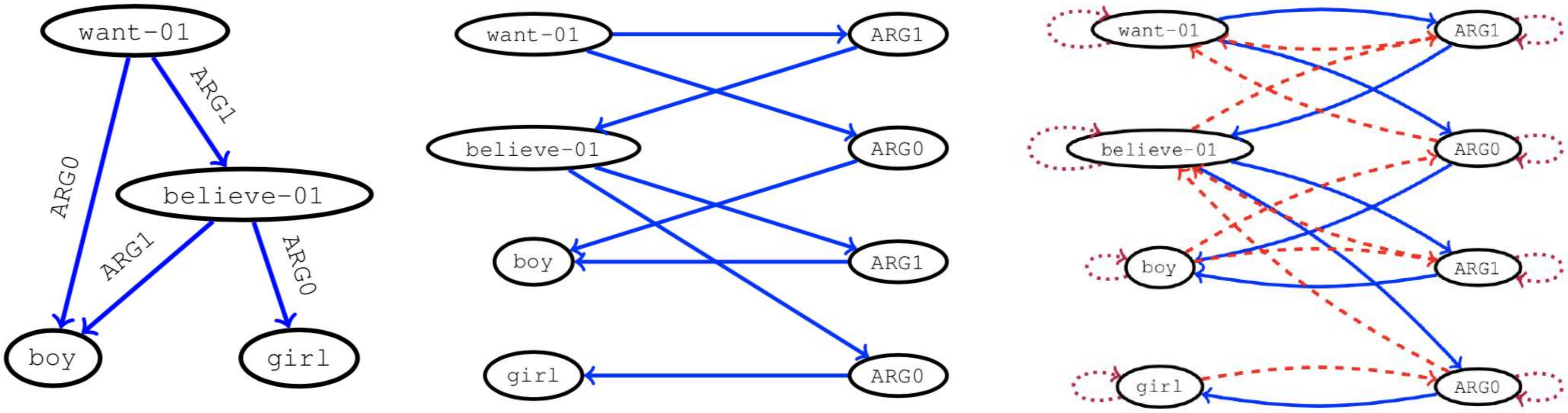


Graph-to-Sequence Learning using Gated Graph Neural Networks ACL 18

AMR-to-Text

Levi Graph Transformation

- Ideally, edges should have instance-specific hidden states
- Transform the input graph into its equivalent Levi graph



{default, reverse, self }

Graph-to-Sequence Learning using Gated Graph Neural Networks ACL 18

AMR-to-Text

$$\mathcal{G} = \{\mathcal{V}, \mathcal{E}, L_{\mathcal{V}}, L_{\mathcal{E}}\}$$

$$\mathbf{h}_v^0 = \mathbf{x}_v$$

reset

$$\mathbf{r}_v^t = \sigma\left(c_v^r \sum_{u \in \mathcal{N}_v} \mathbf{W}_{\boxed{\ell_e}}^r \mathbf{h}_u^{(t-1)} + \mathbf{b}_{\ell_e}^r\right)$$

edge-wise parameters

update

$$\mathbf{z}_v^t = \sigma\left(c_v^z \sum_{u \in \mathcal{N}_v} \mathbf{W}_{\ell_e}^z \mathbf{h}_u^{(t-1)} + \mathbf{b}_{\ell_e}^z\right)$$

$$\widetilde{\mathbf{h}}_v^t = \rho\left(c_v \sum_{u \in \mathcal{N}_v} \mathbf{W}_{\ell_e}\left(\mathbf{r}_u^t \odot \mathbf{h}_u^{(t-1)}\right) + \mathbf{b}_{\ell_e}\right)$$

$$\mathbf{h}_v^t = (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{(i-1)} + \mathbf{z}_v^t \odot \widetilde{\mathbf{h}}_v^t$$

Graph-to-Sequence Learning using Gated Graph Neural Networks ACL 18

AMR-to-Text

- Transforms the input graph into its equivalent **Levi graph**
- Graph Convolutional Network Encoders

$$h_i^{(k+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} W_{\mathrm{dir}(j,i)}^{(k)} h_j^{(k)} + b^{(k)}\right)$$

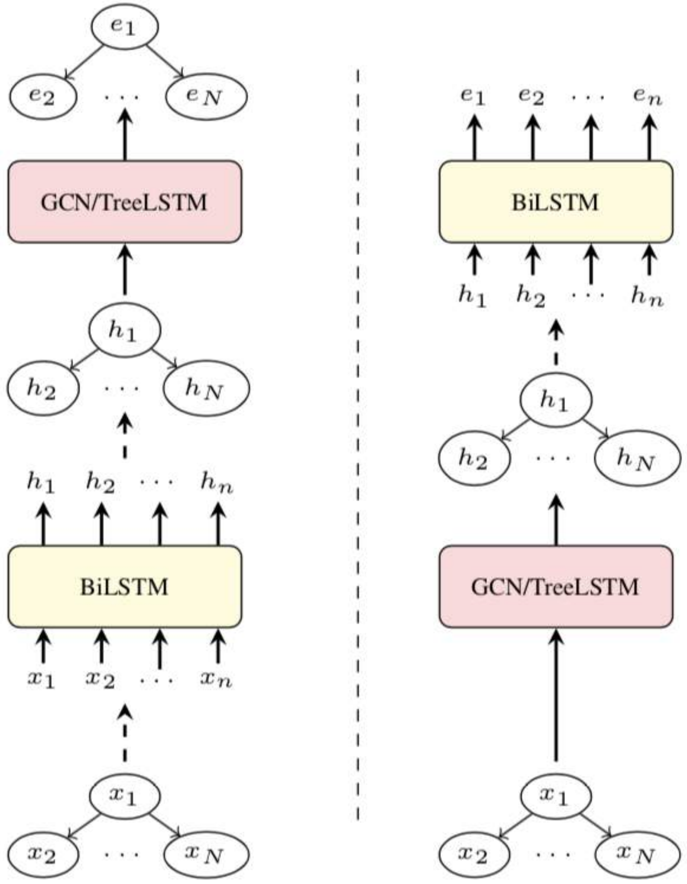
$$e_{1:N} = h_1^{(K)}, \ldots, h_N^{(K)},$$

dir(j, i) indicates the direction of the edge between x_j and x_i

Structural Neural Encoders for AMR-to-text Generation NAACL 19

AMR-to-Text

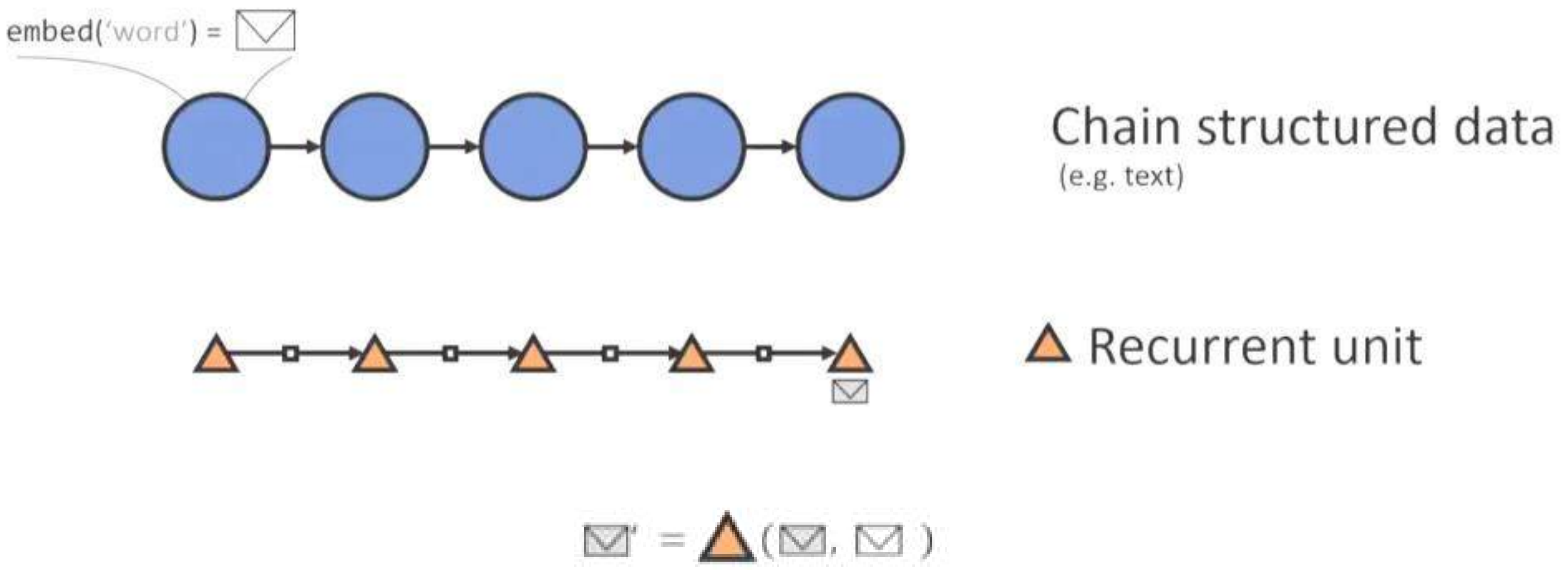
- Stacking Encoders



Structural Neural Encoders for AMR-to-text Generation NAACL 19

AMR-to-Text

- AMR is naturally a Graph.
- However, Text based NLP:



Graph neural networks: Variations and applications

GNN IN NLP

- **AMR-To-Text**
  - A Graph-to-Sequence Model for AMR-to-Text Generation ***ACL 18***
  - Graph-to-Sequence Learning using Gated Graph Neural Networks ***ACL 18***
  - Structural Neural Encoders for AMR-to-text Generation ***NAACL 19***
- **SQL-To-Text**
  - SQL-to-Text Generation with Graph-to-Sequence Model ***EMNLP18***
- **Document Summarization**
  - Structured Neural Summarization ***ICLR 19***
  - Graph-based Neural Multi-Document Summarization ***CoNLL 17***

SQL-to-Text

- SQL-to-text task is to automatically generate human-like descriptions interpreting the meaning of a given structured query language (SQL) query .

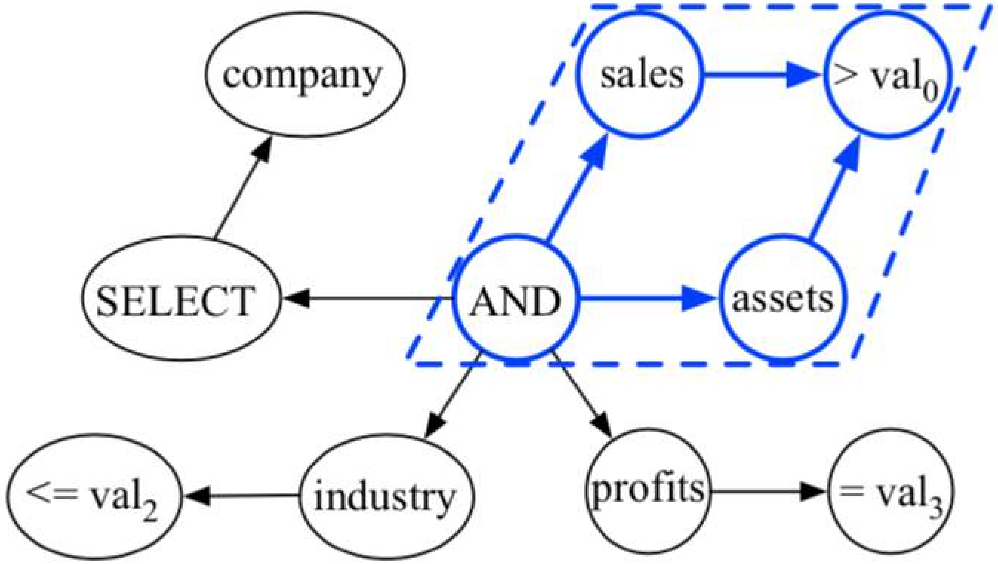
(SQL): **SELECT** company **WHERE** assets > val_0 **AND** sales > val_0 **AND** industry_rank <= val_2 **AND** revenue = val_3

Interpretation:
which company has both the market value and assets higher than $val_{0,}$ ranking in top val_2 and revenue of val_3

SQL-to-Text Generation with Graph-to-Sequence Model EMNLP18

SQL-to-Text

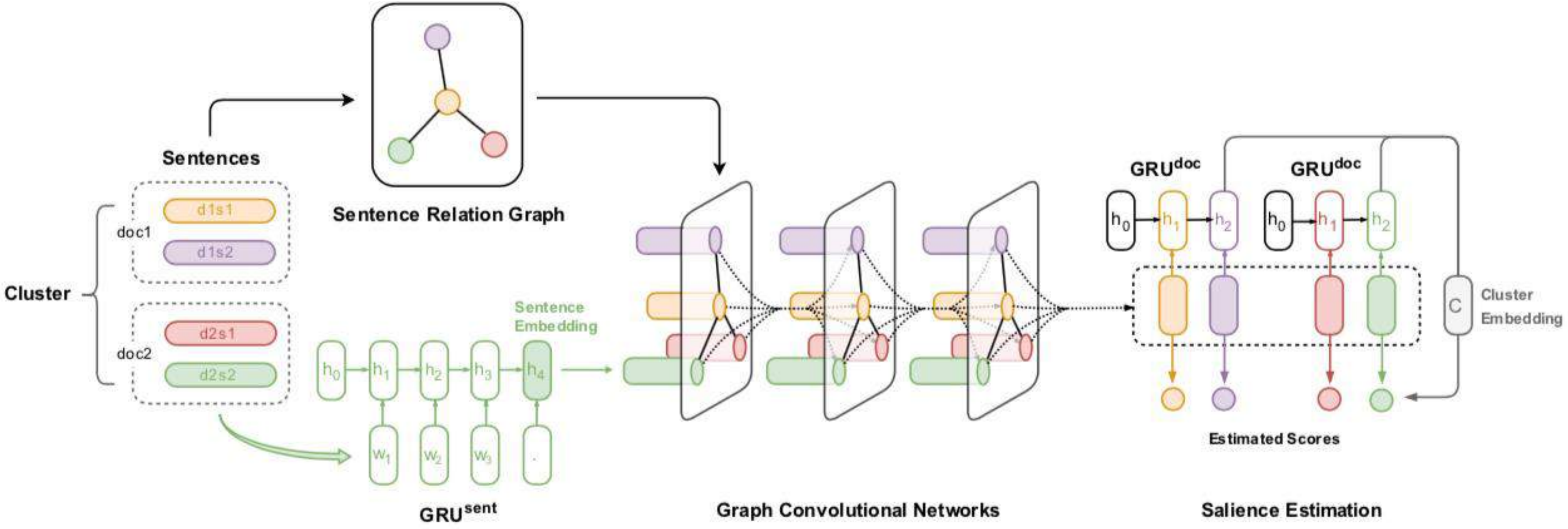
- **Motivation**: representing SQL as a graph instead of a sequence could help the model to better learn the correlation between this graph pattern and the interpretation “...both X and Y higher than Z...”
- SELECT Clause + WHERE Clause.



SQL-to-Text Generation with Graph-to-Sequence Model EMNLP18

Summarization

- Task: Multi-Document Summarization(MDS)



Graph-based Neural Multi-Document Summarization CoNLL 2017

Summarization

- **Cosine similarity**
  - BoW: frequency based
  - Threshold > 0.2
  - TF-IDF First

- **Approximate Discourse Graph (ADG).**
  - The ADG constructs edges between sentences by counting discourse relation indicators such as deverbal noun references, event / entity continuations, discourse markers, and coreferent mentions. These features allow characterization of sentence relationships, rather than simply their similarity.

Graph-based Neural Multi-Document Summarization CoNLL 2017

Summarization

- Input

$A \in \mathbb{R}^{N \times N}$ adjacency matrix

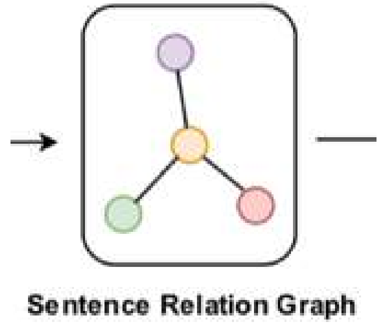
$X \in \mathbb{R}^{N \times D}$ input node feature matrix

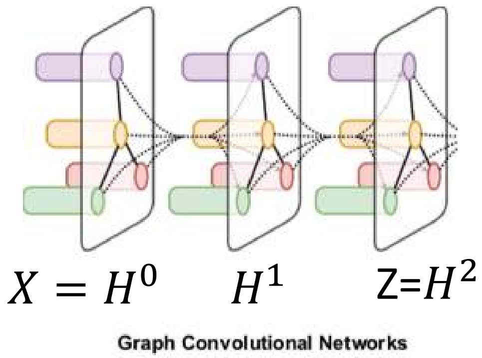
- Output

$Z \in \mathbb{R}^{N \times F}$ high-level hidden features for each node

$$H^{(l+1)} = \sigma\left(AH^{(l)}W^{(l)}\right)$$

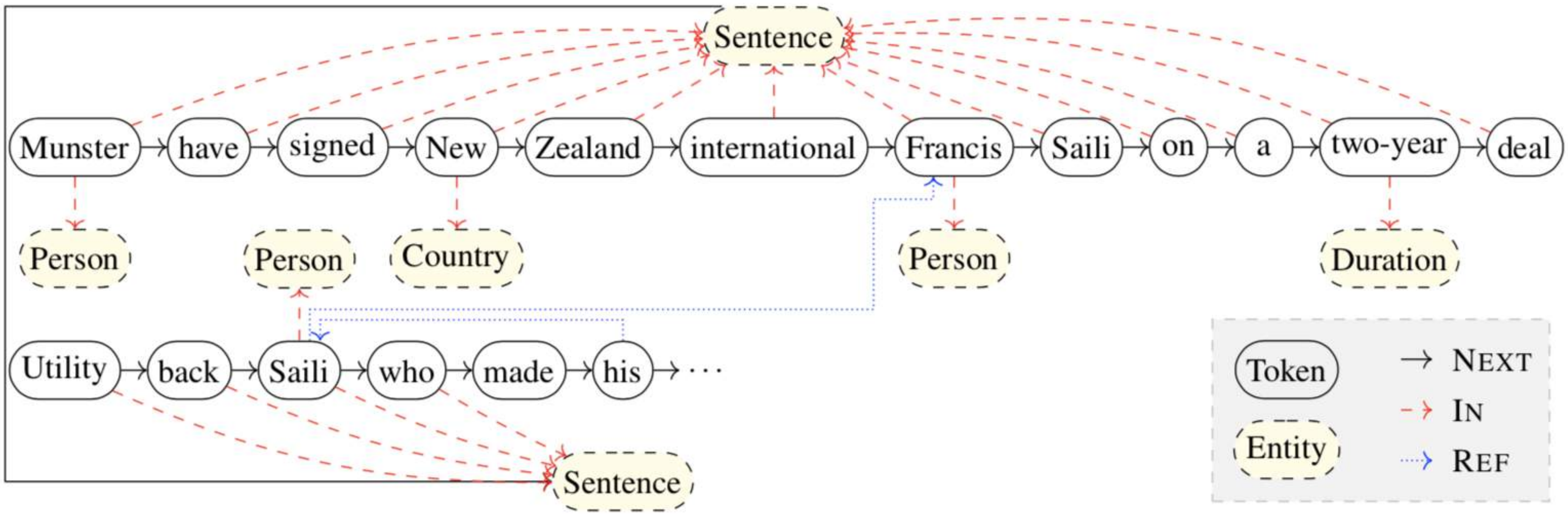
$$Z = f(X, A) = H^{(L)}$$





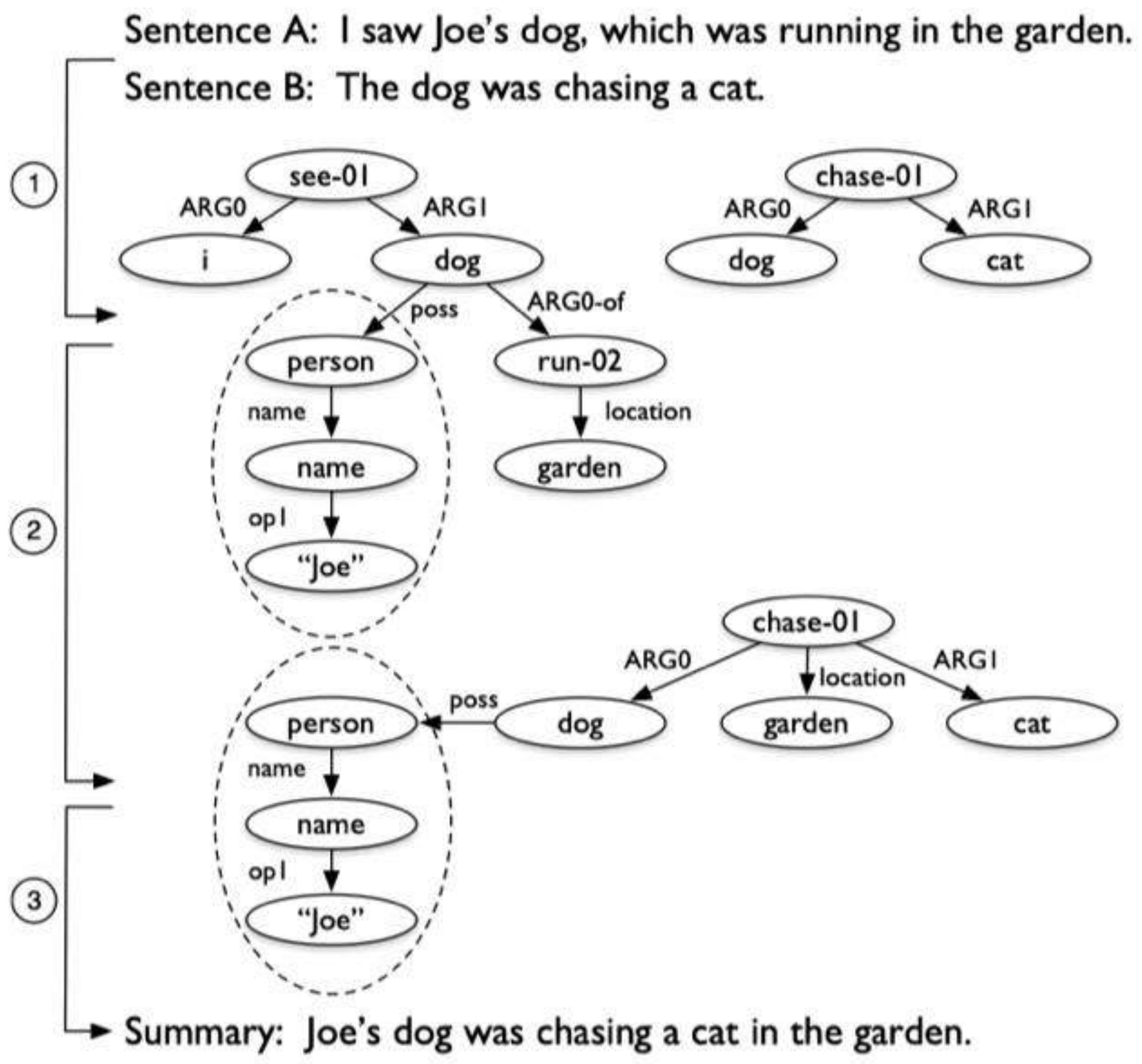
Graph-based Neural Multi-Document Summarization CoNLL 2017

Summarization



Structured Neural Summarization ICLR 19

Summarization && AMR



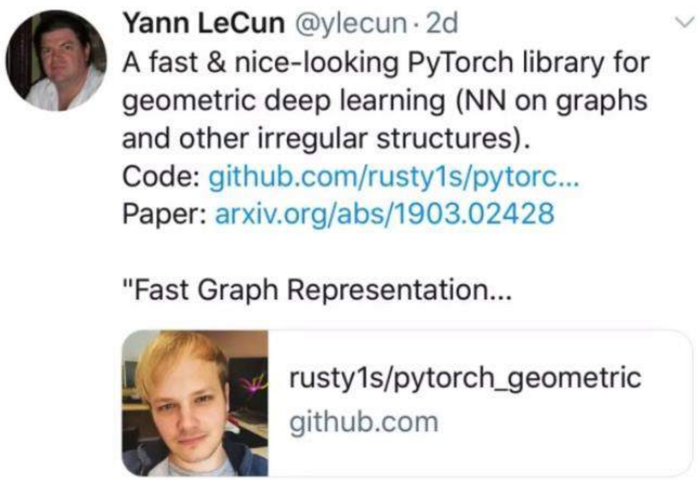
Toward Abstractive Summarization Using Semantic Representations NAACL15

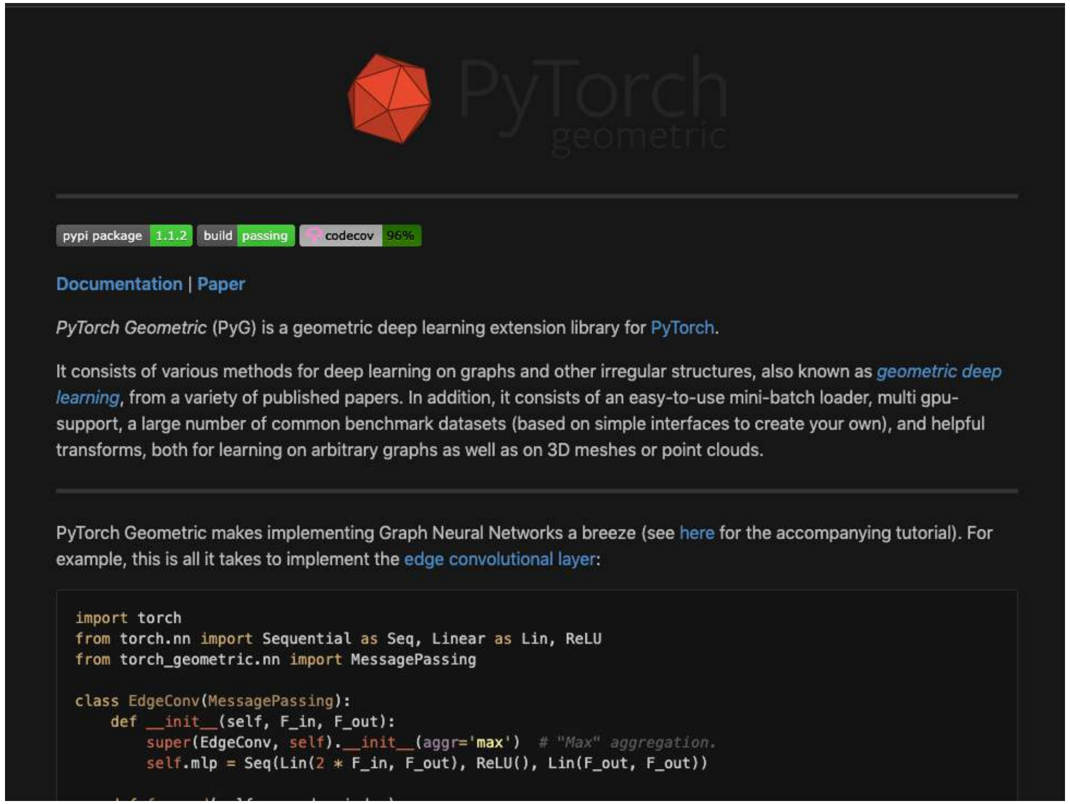
Outline

1. Basic && Overview
2. Graph Neural Networks
    1. Original Graph Neural Networks (GNNs)
    2. Graph Convolutional Networks (GCNs) && Graph SAGE
    3. Gated Graph Neural Networks (GGNNs)
    4. Graph Neural Networks With Attention (GAT)
    5. Sub-Graph Embeddings
3. Message Passing Neural Networks (MPNN)
4. GNN In NLP (AMR、SQL、Summarization)
5. Tools
6. Conclusion

Tools

- https://github.com/rusty1s/pytorch_geometric
- https://github.com/dmlc/dgl





Documentation | Paper

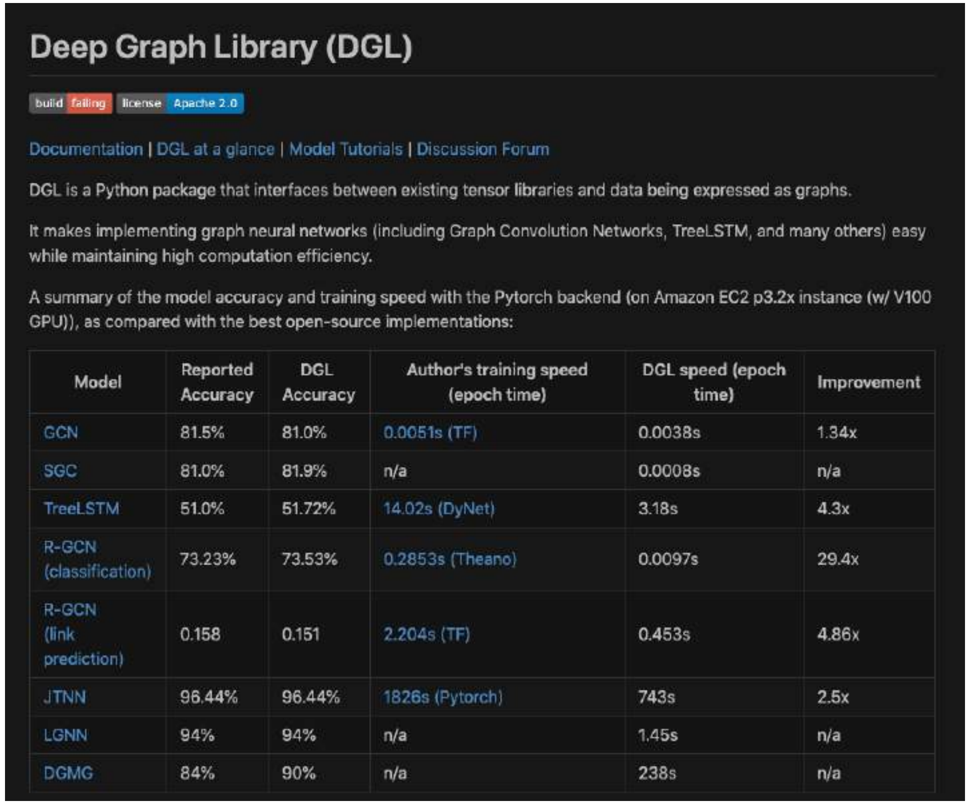
PyTorch Geometric (PyG) is a geometric deep learning extension library for PyTorch.

It consists of various methods for deep learning on graphs and other irregular structures, also known as *geometric deep learning*, from a variety of published papers. In addition, it consists of an easy-to-use mini-batch loader, multi gpu-support, a large number of common benchmark datasets (based on simple interfaces to create your own), and helpful transforms, both for learning on arbitrary graphs as well as on 3D meshes or point clouds.

PyTorch Geometric makes implementing Graph Neural Networks a breeze (see here for the accompanying tutorial). For example, this is all it takes to implement the edge convolutional layer:

```
import torch
from torch.nn import Sequential as Seq, Linear as Lin, ReLU
from torch_geometric.nn import MessagePassing

class EdgeConv(MessagePassing):
    def __init__(self, F_in, F_out):
        super(EdgeConv, self).__init__(aggr='max')  # "Max" aggregation.
        self.mlp = Seq(Lin(2 * F_in, F_out), ReLU(), Lin(F_out, F_out))
```



Deep Graph Library (DGL)

Documentation | DGL at a glance | Model Tutorials | Discussion Forum

DGL is a Python package that interfaces between existing tensor libraries and data being expressed as graphs.

It makes implementing graph neural networks (including Graph Convolution Networks, TreeLSTM, and many others) easy while maintaining high computation efficiency.

A summary of the model accuracy and training speed with the Pytorch backend (on Amazon EC2 p3.2x instance (w/ V100 GPU)), as compared with the best open-source implementations:

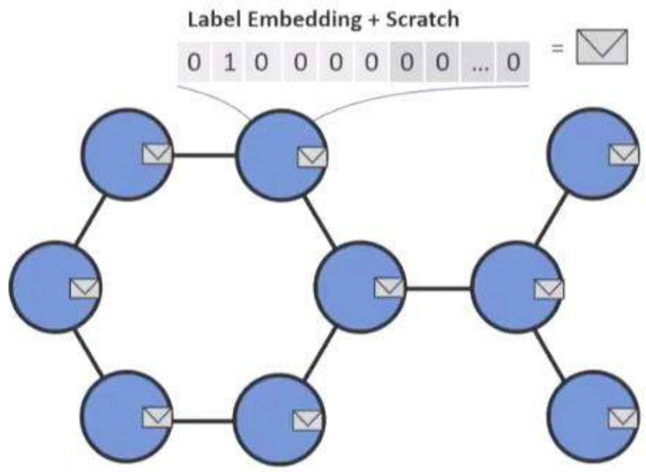
Model	Reported Accuracy	DGL Accuracy	Author's training speed (epoch time)	DGL speed (epoch time)	Improvement
GCN	81.5%	81.0%	0.0051s (TF)	0.0038s	1.34x
SGC	81.0%	81.9%	n/a	0.0008s	n/a
TreeLSTM	51.0%	51.72%	14.02s (DyNet)	3.18s	4.3x
R-GCN (classification)	73.23%	73.53%	0.2853s (Theano)	0.0097s	29.4x
R-GCN (link prediction)	0.158	0.151	2.204s (TF)	0.453s	4.86x
JTNN	96.44%	96.44%	1826s (Pytorch)	743s	2.5x
LGNN	94%	94%	n/a	1.45s	n/a
DGMG	84%	90%	n/a	238s	n/a

Outline

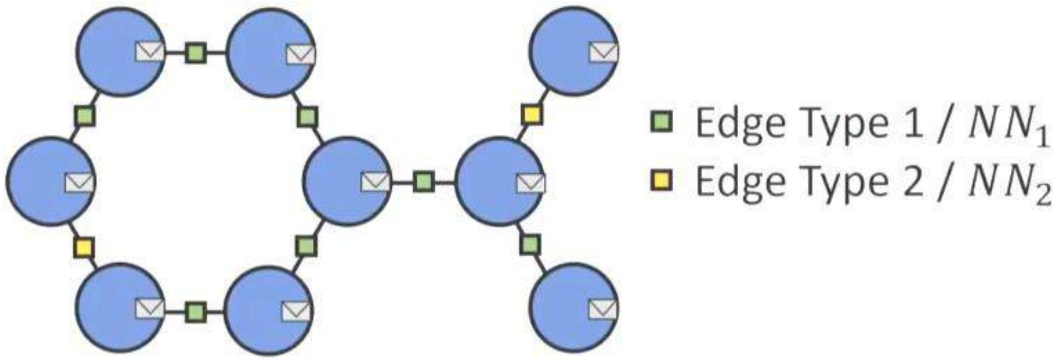
1. Basic && Overview
2. Graph Neural Networks
    1. Original Graph Neural Networks (GNNs)
    2. Graph Convolutional Networks (GCNs) && Graph SAGE
    3. Gated Graph Neural Networks (GGNNs)
    4. Graph Neural Networks With Attention (GAT)
    5. Sub-Graph Embeddings
3. Message Passing Neural Networks (MPNN)
4. GNN In NLP (AMR、SQL、Summarization)
5. Tools
6. Conclusion

Conclusion

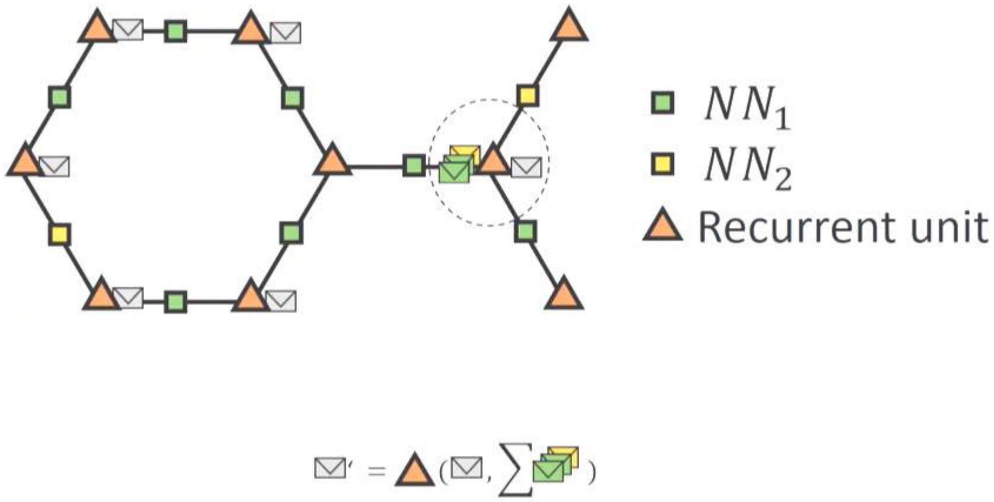
(1)



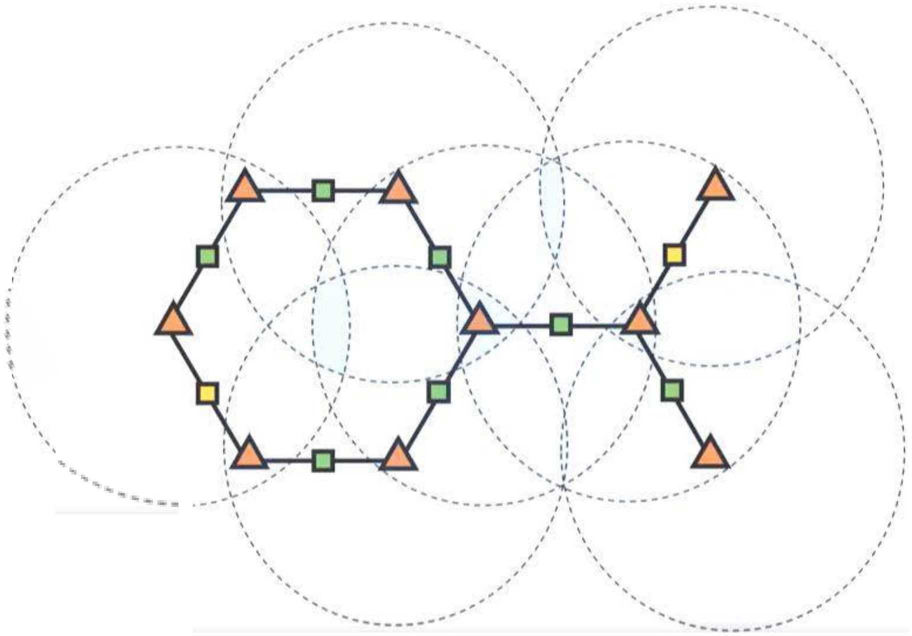
(2)



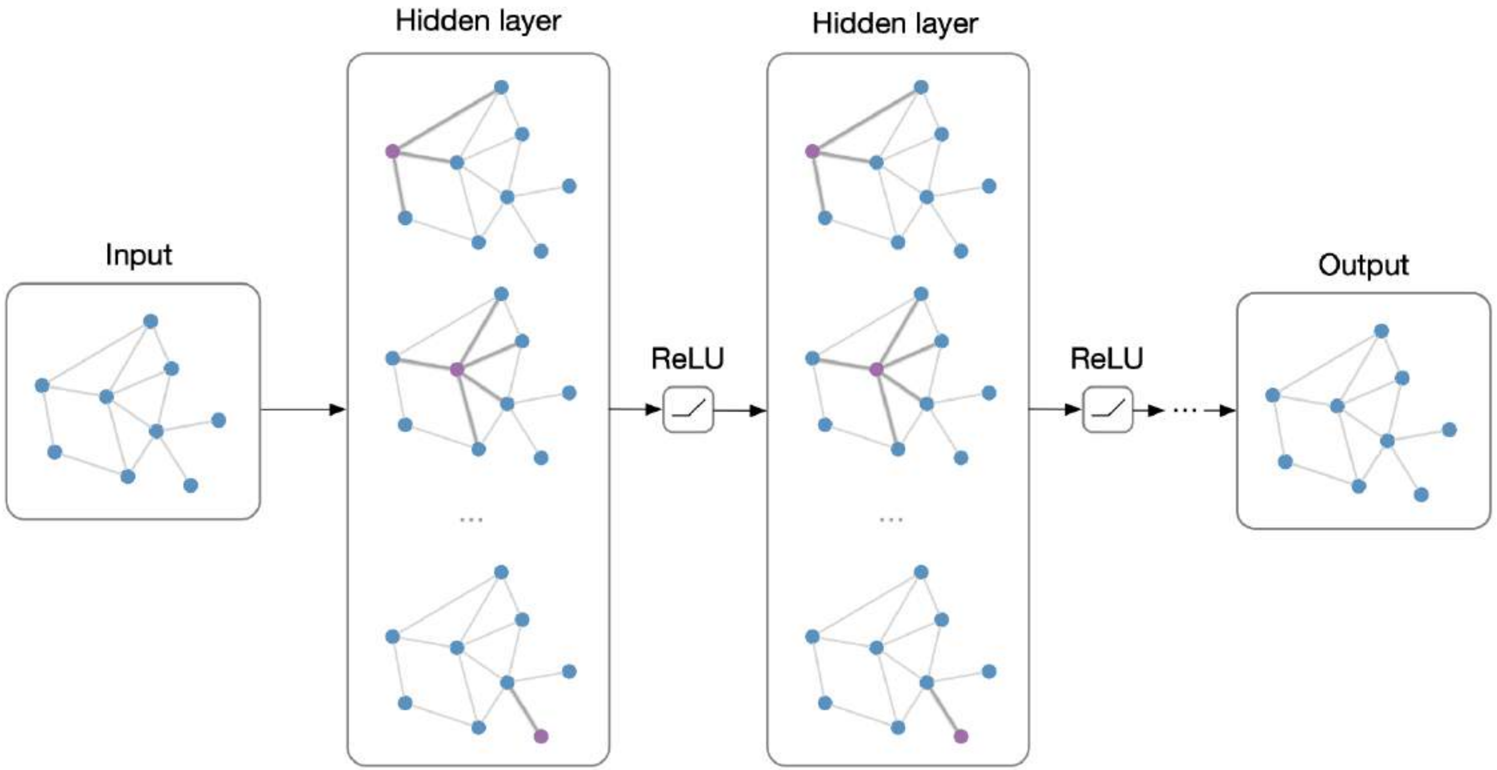
(3)



(4)



Thanks!



Xiachong Feng
TG

2019-04