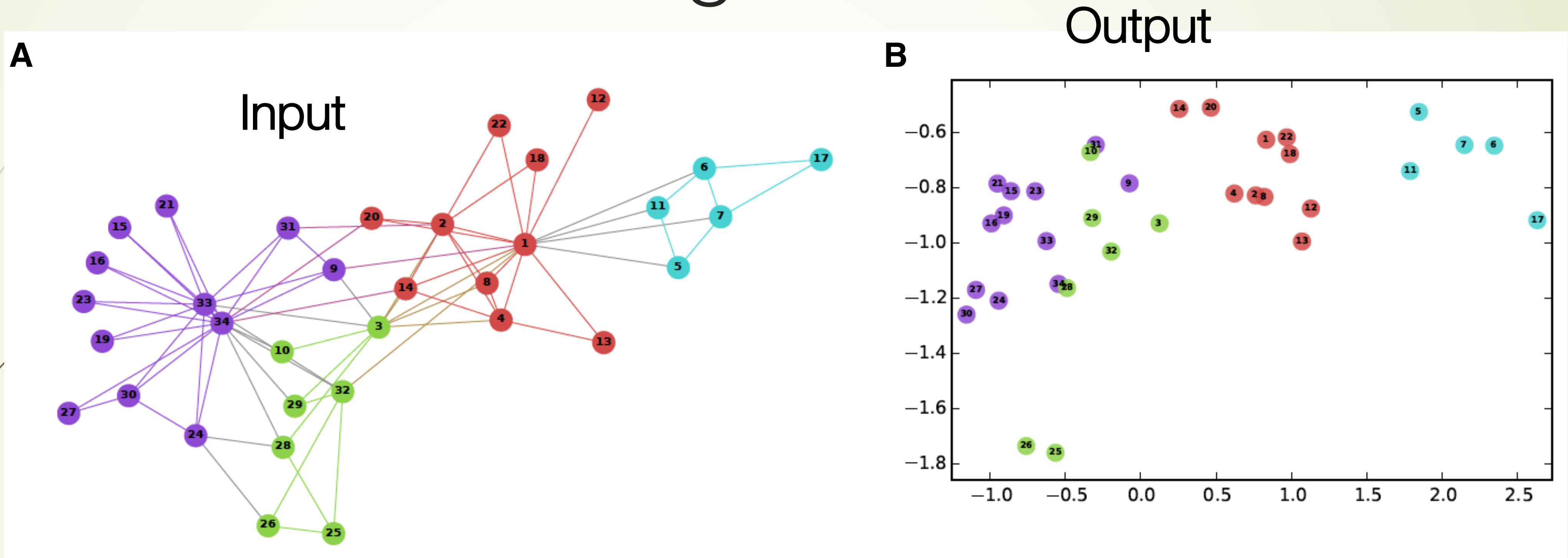
node2vec:
Scalable Feature Learning for Networks

Aditya Grover and Jure Leskovec. KDD 2016.

Presented by Haoxiang Wang. Feb 26, 2020.

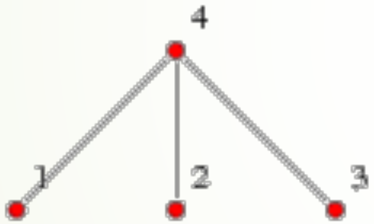
Node Embeddings

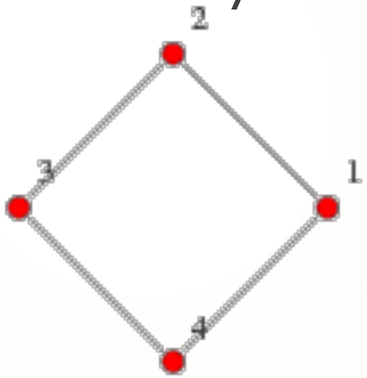


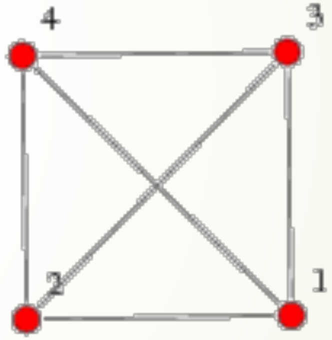
- **Intuition:** Find embeddings of nodes in a d-dimensional space so that "similar" nodes in the graph have embeddings that are close together.

Setup

- Assume we have a graph $\mathbf{G}$:
  - $\mathbf{V}$ is the vertex set (i.e., node set).
  - $\mathbf{A}$ is the adjacency matrix (assume binary).





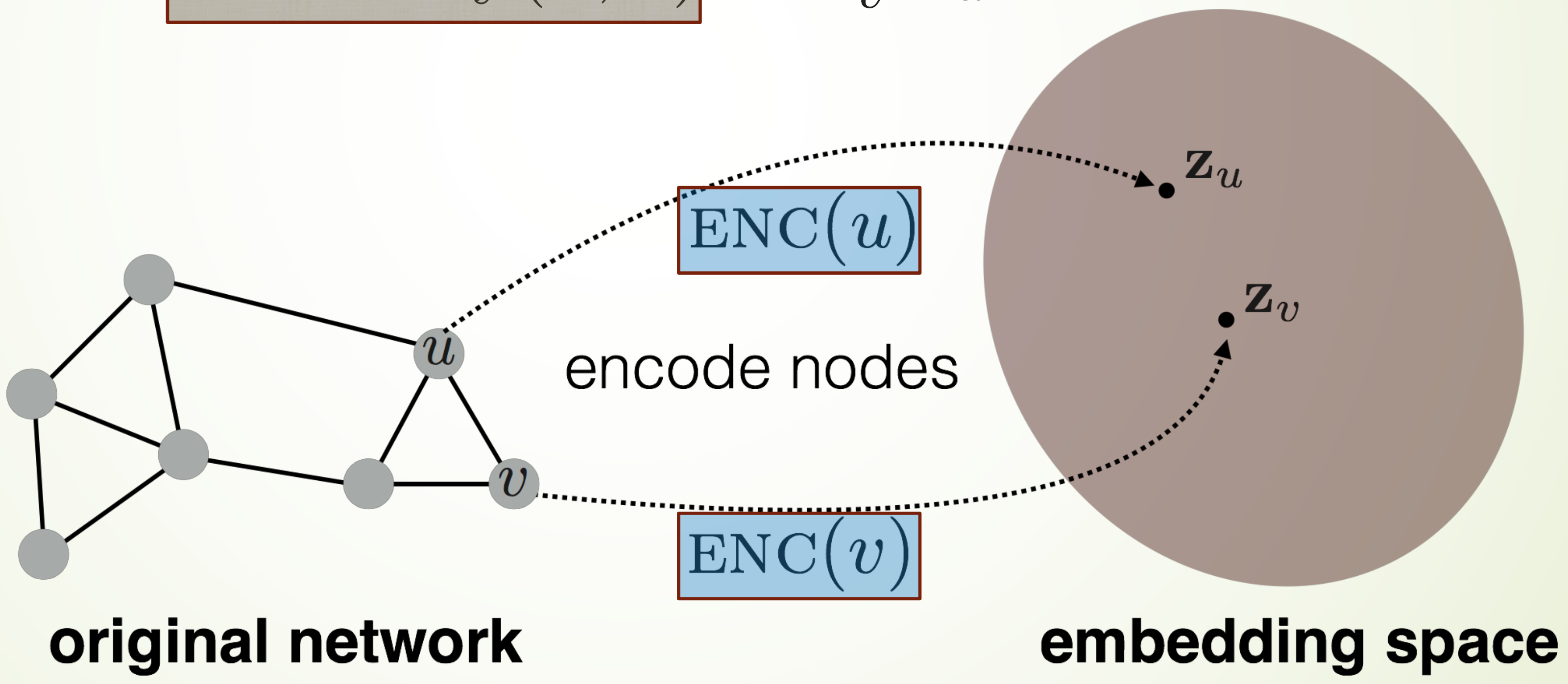


$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Embedding Nodes

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

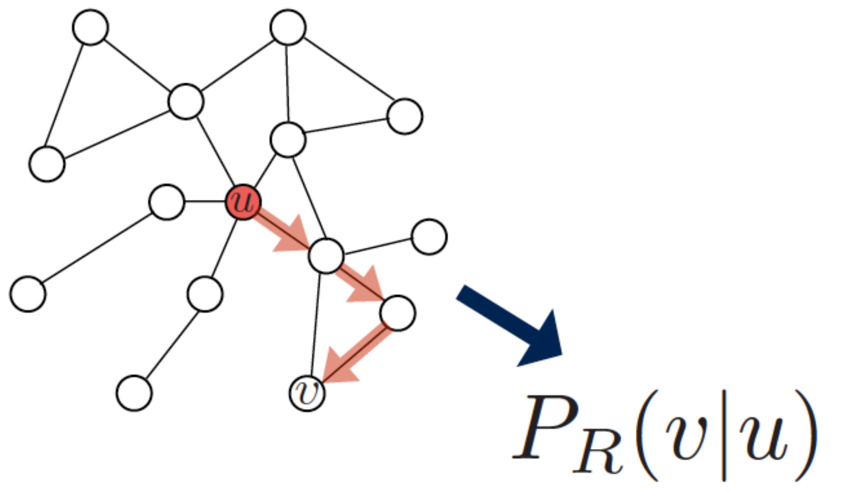
$$\boxed{\text{similarity}(u, v)} \approx \mathbf{z}_v^\top \mathbf{z}_u$$



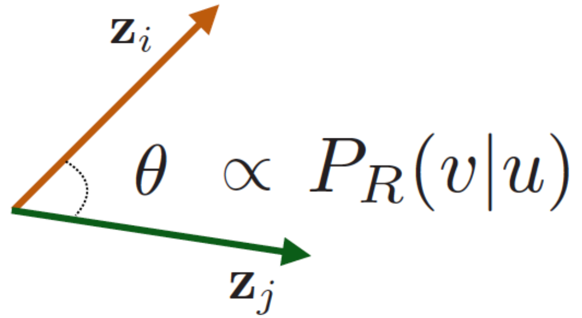
Random Walk Embeddings: Basic Idea

$$\mathbf{z}_u^\top \mathbf{z}_v \approx$$ probability that u and v co-occur on a random walk over the network

1. Estimate probability of visiting node v on a random walk starting from node u using some random walk strategy R.

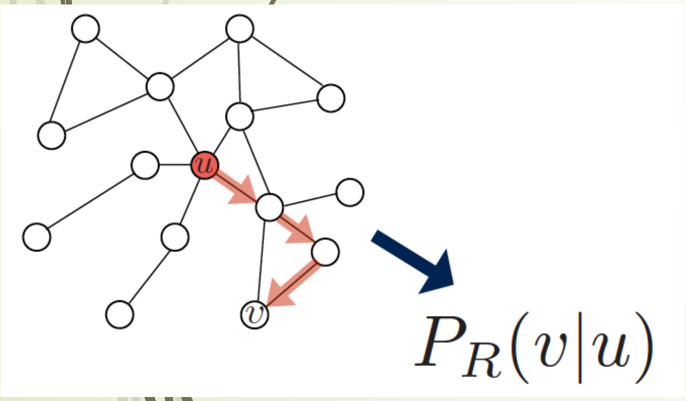


2. Optimize embeddings to encode these random walk statistics.



Algorithm/Optimization of Random Walk Embeddings

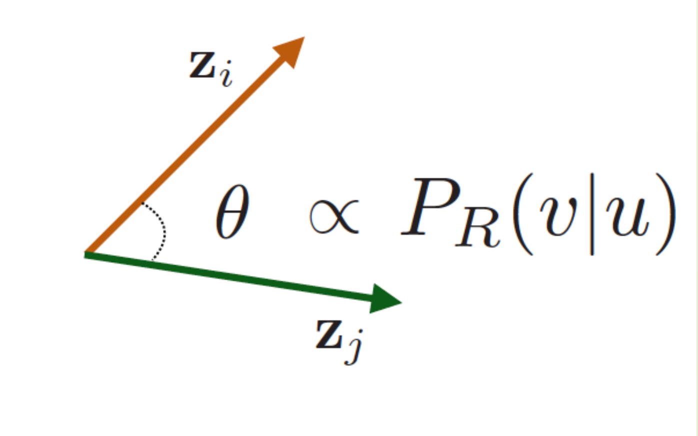
1. Run short random walks starting from each node on the graph using some strategy R.
2. For each node u collect $N_R(u)$, the multiset* of nodes visited on random walks starting from u. (* $N_R(u)$ can have repeat elements since nodes can be visited multiple times on random walks.)
3. Optimize embeddings to according to:



$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

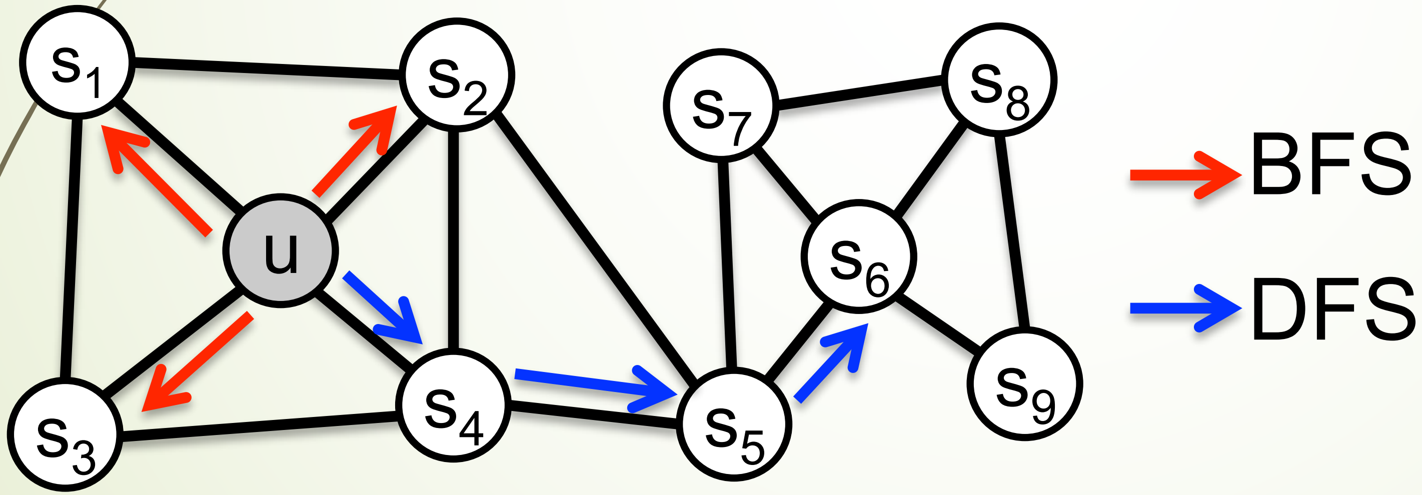
$$P(v|\mathbf{z}_u) = \frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\boxed{\sum_{n \in V}} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}$$

In practice, random sampling based on some distribution over nodes



Node2vec: Biased Random Walks

- **Idea:** use flexible, biased random walks that can trade off between **local** and **global** views of the network (Grover and Leskovec, 2016).
- BFS (Breath-First Search)and DFS (Depth-First Search): Two classic strategies to define a neighborhood $N_R(u)$ of a given node u:



$N_{BFS}(u) = \{ s_1, s_2, s_3\}$

Local microscopic view

$N_{DFS}(u) = \{ s_4, s_5, s_6\}$

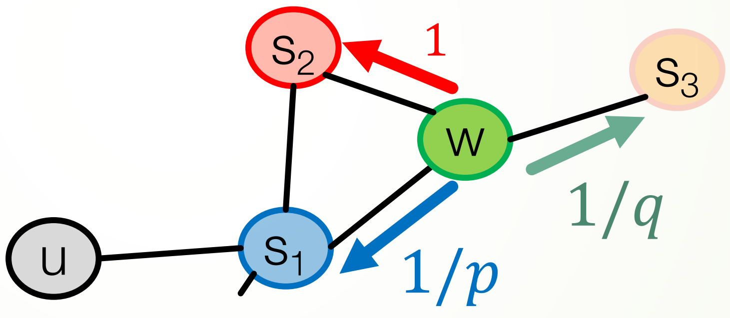
Global macroscopic view

Combine BFS + DFS by a Ratio

Biased random walk R that given a node u generates neighborhood $N_R(u)$

- Two parameters:
  - Return parameter p: Return back to the previous node
  - Walk-away parameter q : Moving outwards (DFS) vs. inwards (BFS)

Walker is at w.
Where to go next?



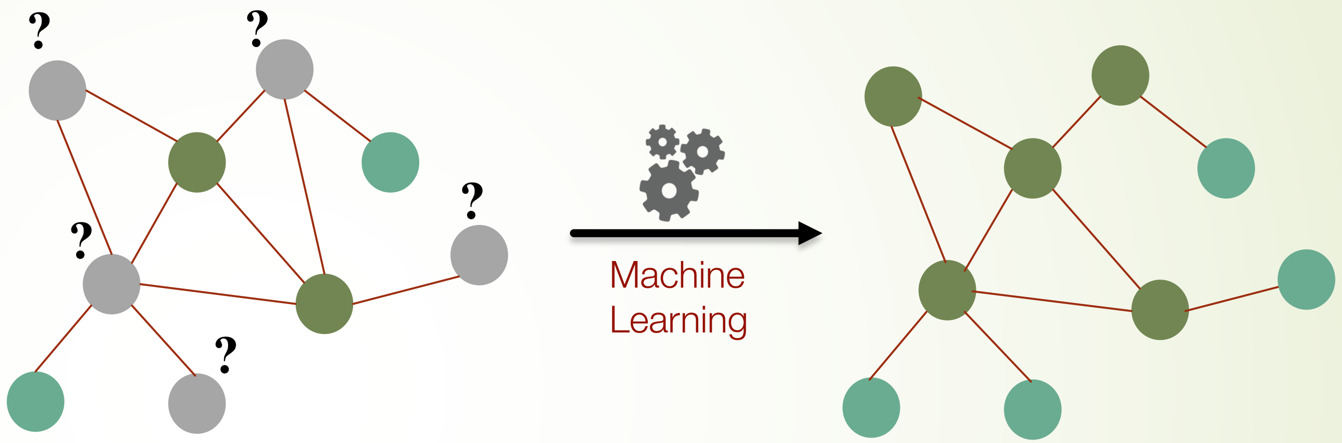
Unnormalized transition prob.

$w \rightarrow$		
	s_1	$1/p$
	s_2	1
	s_3	$1/q$

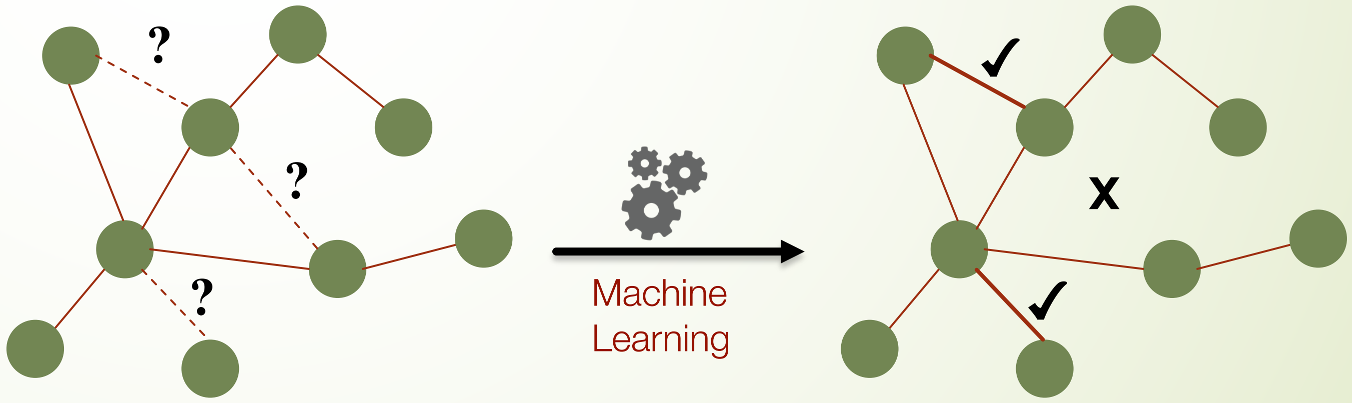
BFS-like walk: Low value of p
DFS-like walk: Low value of q

Benchmarks: Node Classification & Link Prediction

Node Classification



Link Prediction



Empirical Results

Node Classification

Algorithm	BlogCatalog	PPI	Wikipedia
	Dataset		
Spectral Clustering	0.0405	0.0681	0.0395
DeepWalk	0.2110	0.1768	0.1274
LINE	0.0784	0.1447	0.1164
node2vec	**0.2581**	**0.1791**	**0.1552**
node2vec settings (p,q)	0.25, 0.25	4, 1	4, 0.5
Gain of *node2vec* [%]	**22.3**	**1.3**	**21.8**

Table 2: Macro-F_1 scores for multilabel classification on BlogCatalog, PPI (Homo sapiens) and Wikipedia word cooccurrence networks with 50% of the nodes labeled for training.

Link Prediction

Op	Algorithm	Facebook	PPI	arXiv
		Dataset		
	Common Neighbors	0.8100	0.7142	0.8153
	Jaccard's Coefficient	0.8880	0.7018	0.8067
	Adamic-Adar	0.8289	0.7126	0.8315
	Pref. Attachment	0.7137	0.6670	0.6996
(a)	Spectral Clustering	0.5960	0.6588	0.5812
	DeepWalk	0.7238	0.6923	0.7066
	LINE	0.7029	0.6330	0.6516
	node2vec	0.7266	0.7543	0.7221
(b)	Spectral Clustering	0.6192	0.4920	0.5740
	DeepWalk	**0.9680**	0.7441	0.9340
	LINE	0.9490	0.7249	0.8902
	node2vec	**0.9680**	**0.7719**	**0.9366**
(c)	Spectral Clustering	0.7200	0.6356	0.7099
	DeepWalk	0.9574	0.6026	0.8282
	LINE	0.9483	0.7024	0.8809
	node2vec	0.9602	0.6292	0.8468
(d)	Spectral Clustering	0.7107	0.6026	0.6765
	DeepWalk	0.9584	0.6118	0.8305
	LINE	0.9460	0.7106	0.8862
	node2vec	0.9606	0.6236	0.8477

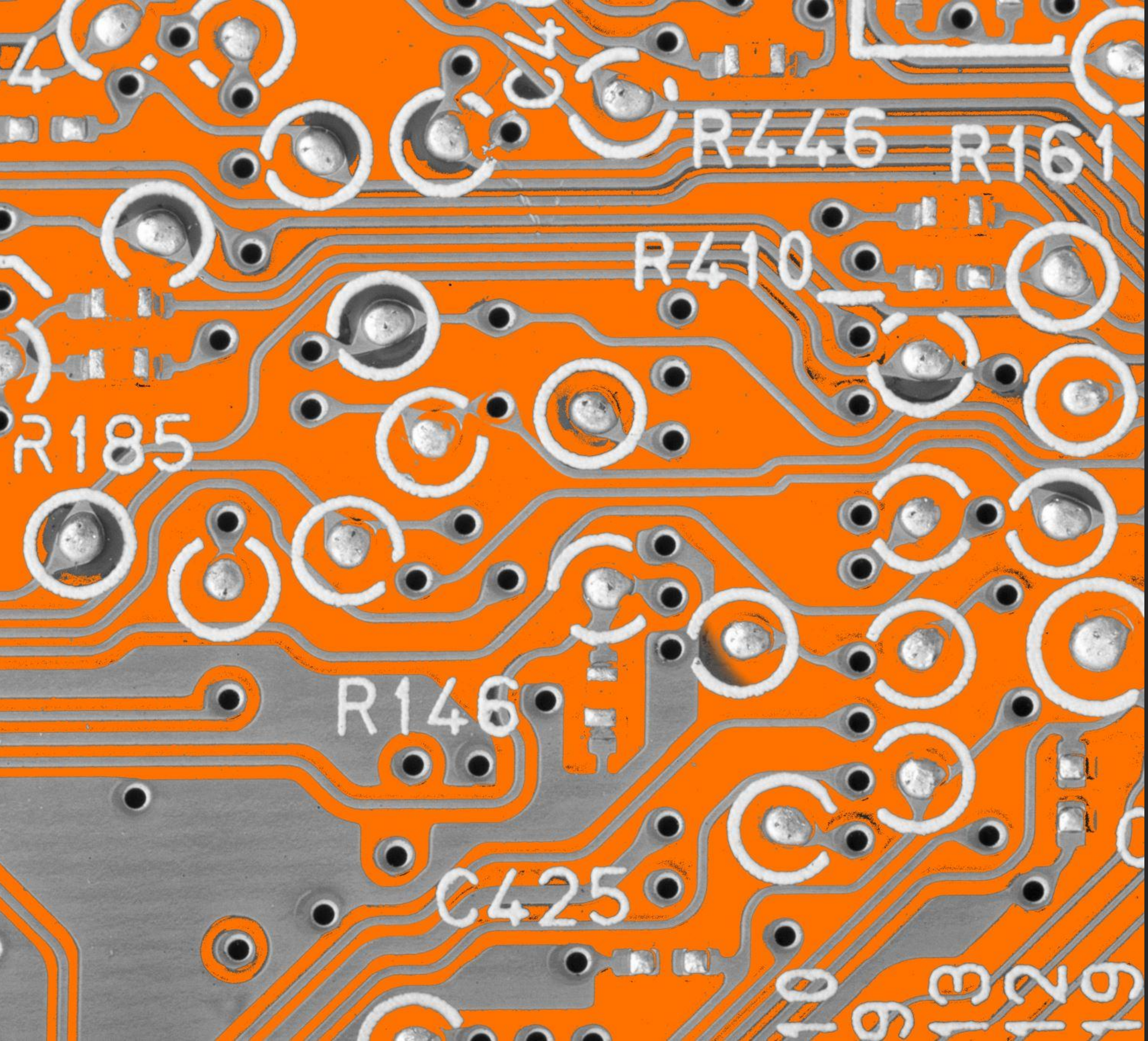
Table 4: Area Under Curve (AUC) scores for link prediction. Comparison with popular baselines and embedding based methods bootstapped using binary operators: (a) Average, (b) Hadamard, (c) Weighted-L1, and (d) Weighted-L2 (See Table 1 for definitions).

Advantages of Node2Vec

- node2vec performs better on **node classification** compared with other node embedding methods.
- Random walk approaches are generally more efficient (i.e., $\mathrm{O}(|\mathrm{E}|)$ vs. $\mathrm{O}(|\mathrm{V}|^2)$)
- (Note: In general, one must choose definition of node similarity that matches application.)

Other random walk node embedding works

- **Different kinds of biased random walks:**
  - Based on node attributes (Dong et al., 2017).
  - Based on a learned weights (Abu-El-Haija et al., 2017)
- **Alternative optimization schemes:**
  - Directly optimize based on 1-hop and 2-hop random walk probabilities (as in LINE from Tang et al. 2015).
- **Network preprocessing techniques:**
  - Run random walks on modified versions of the original network (e.g., Ribeiro et al. 2017's struct2vec, Chen et al. 2016's HARP).

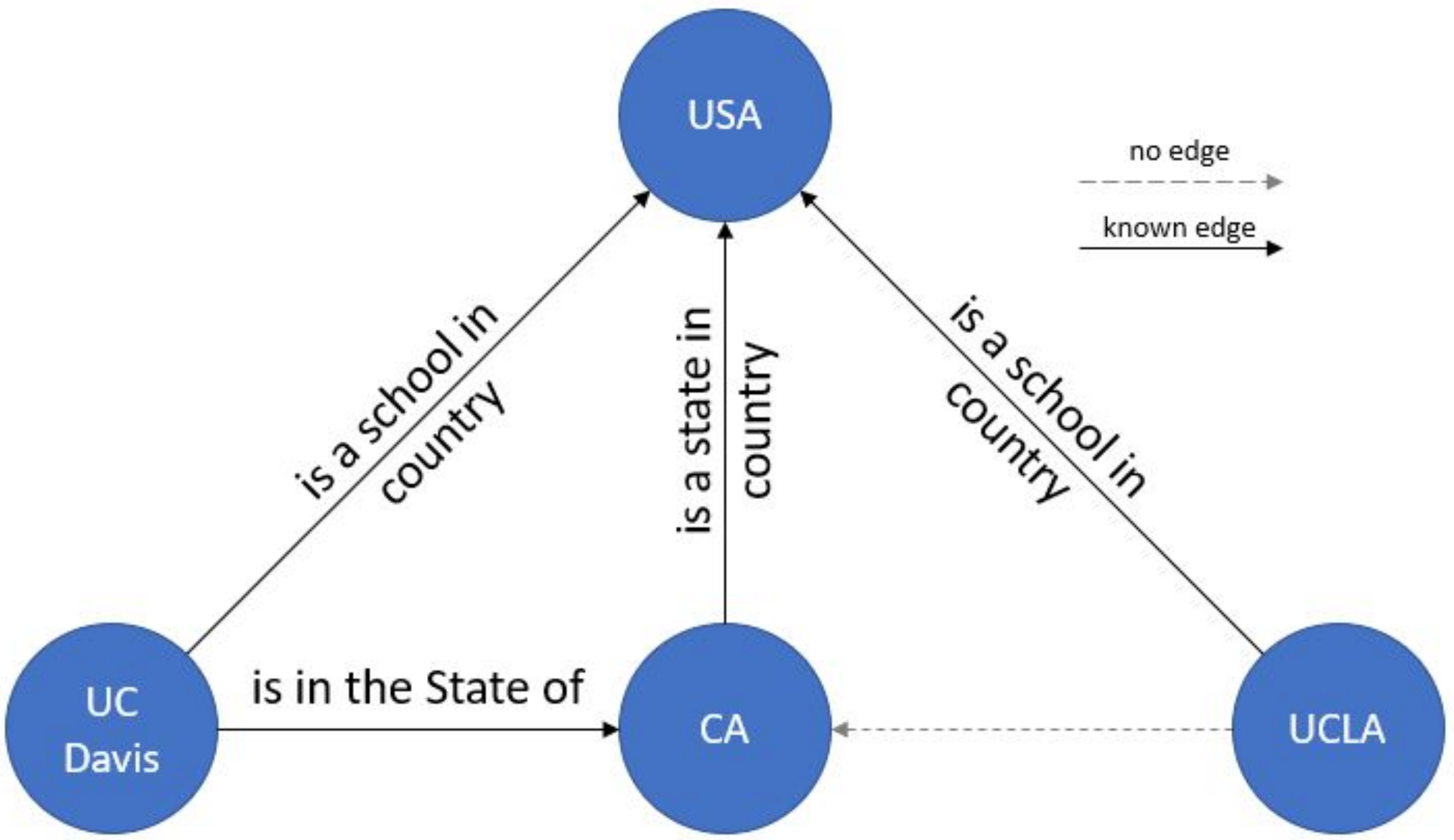


LEARNING ENTITY AND RELATION EMBEDDINGS FOR KNOWLEDGE GRAPH COMPLETION

XIAODAN DU

KNOWLEDGE GRAPH COMPLETION

- Predicting relations between entities under supervision of the existing knowledge graph



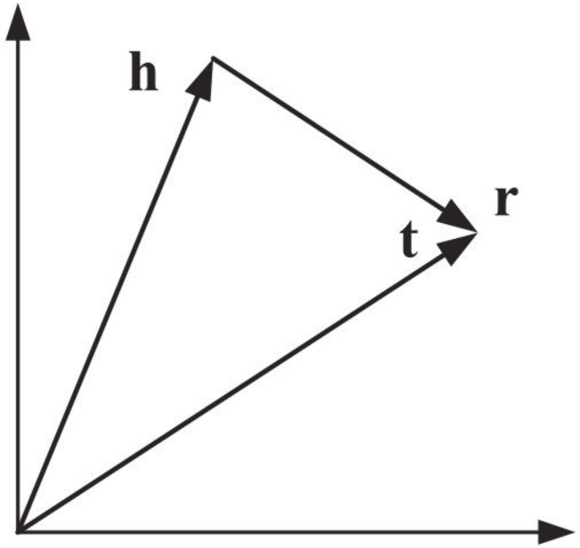
KNOWLEDGE GRAPH EMBEDDING

- Embedding a knowledge graph into a continuous vector space while preserving certain information of the graph
- Learning vector embeddings for both entities and relationships
  - TransE (Bordes et al. 2013), TransH (Wang et al. 2014): assume embeddings of entities and relations belong to a single space $\mathbb{R}^k$
  - TransR: assumes one **entity space** and multiple **relation spaces**

TRANSE AND TRANSH

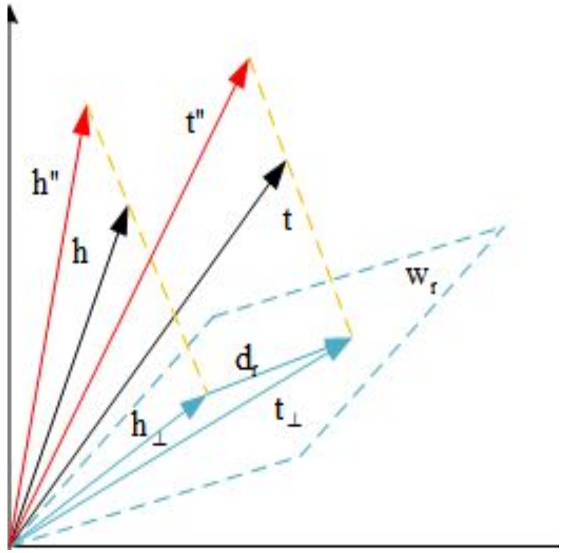
IF TRIPLE (H, R, T) HOLDS

TransE



$$f_r(h,t) = \|\mathbf{h} + \mathbf{r} - \mathbf{t}\|_2^2$$

TransH



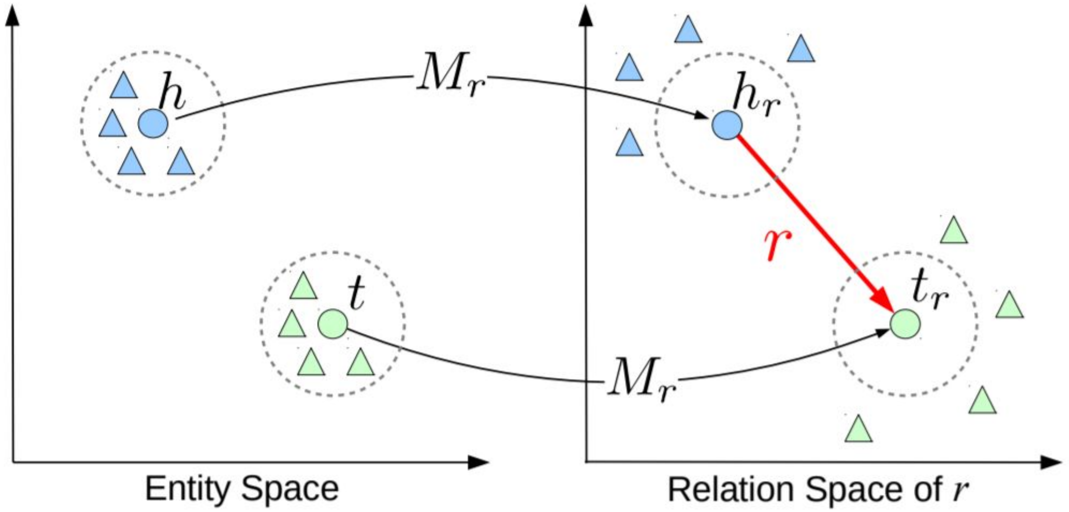
$$f_r(h,t) = \|\mathbf{h}_\perp + \mathbf{r} - \mathbf{t}_\perp\|_2^2.$$

$$\mathbf{h}_\perp = \mathbf{h} - \mathbf{w}_r^\top \mathbf{h} \mathbf{w}_r$$

Solves the problem of 1-to-N, N-to-1 and N-to-N relations

TRANSR

- Authors argue that:
  - relations and entities are completely different objects, so they shouldn't be embedded in the same semantic space.
  - Even though TransH extends modeling flexibility, it **does not** perfectly break the restrict of a common semantic space



h, t $\in \mathbb{R}^k$; r $\in \mathbb{R}^d$ $\quad$ $M_r \in \mathbb{R}^{k \times d}$

$$\mathbf{h}_r = \mathbf{h}\mathbf{M}_r, \quad \mathbf{t}_r = \mathbf{t}\mathbf{M}_r.$$

$$f_r(h,t) = \|\mathbf{h}_r + \mathbf{r} - \mathbf{t}_r\|_2^2.$$

CTRANSR – CLUSTER-BASED TRANSR

A UNIQUE VECTOR FOR EACH RELATION MIGHT BE UNDER-REPRESENTATIVE

	⟨Head, Tail⟩
1	⟨Africa, Congo⟩, ⟨Asia, Nepal⟩, ⟨Americas, Aruba⟩, ⟨Oceania, Federated States of Micronesia⟩
2	⟨United States of America, Kankakee⟩, ⟨England, Bury St Edmunds⟩, ⟨England, Darlington⟩, ⟨Italy, Perugia⟩
3	⟨Georgia, Chatham County⟩, ⟨Idaho, Boise⟩, ⟨Iowa, Polk County⟩, ⟨Missouri, Jackson County⟩, ⟨Nebraska, Cass County⟩
4	⟨Sweden, Lund University⟩, ⟨England, King's College at Cambridge⟩, ⟨Fresno, California State University at Fresno⟩, ⟨Italy, Milan Conservatory⟩

Basic idea of CTransR: Grouping head-tail pairs into different clusters and learning relation embeddings for each cluster

CTRANSR – CLUSTER-BASED TRANSR

A UNIQUE VECTOR FOR EACH RELATION MIGHT BE UNDER-REPRESENTATIVE

1. Obtain entity embeddings h and t for all (*h*, *t*) pairs using TransE
2. Compute vector offsets (h - t) for all training data for each relation *r*
3. Vector offsets for a certain relation are likely to form multiple clusters
4. Learn a separate relation vector r_c for each cluster and matrix M_r for each relation, respectively (Authors seem to assume different clusters within the same relation share a single relation space)

$$\mathbf{h}_{r,c} = \mathbf{h}\mathbf{M}_r \quad \mathbf{t}_{r,c} = \mathbf{t}\mathbf{M}_r$$

$$f_r(h,t) = \|\mathbf{h}_{r,c} + \mathbf{r}_c - \mathbf{t}_{r,c}\|_2^2 + \alpha\|\mathbf{r}_c - \mathbf{r}\|_2^2,$$

EXPERIMENT RESULTS

Link Prediction: predicting the missing *h* or *t* for a relation fact triple (*h*, *r*, *t*)

Data Sets	WN18				FB15K			
Metric	Mean Rank		Hits@10 (%)		Mean Rank		Hits@10 (%)	
	Raw	Filter	Raw	Filter	Raw	Filter	Raw	Filter
Unstructured (Bordes et al. 2012)	315	304	35.3	38.2	1,074	979	4.5	6.3
RESCAL (Nickel, Tresp, and Kriegel 2011)	1,180	1,163	37.2	52.8	828	683	28.4	44.1
SE (Bordes et al. 2011)	1,011	985	68.5	80.5	273	162	28.8	39.8
SME (linear) (Bordes et al. 2012)	545	533	65.1	74.1	274	154	30.7	40.8
SME (bilinear) (Bordes et al. 2012)	526	509	54.7	61.3	284	158	31.3	41.3
LFM (Jenatton et al. 2012)	469	456	71.4	81.6	283	164	26.0	33.1
TransE (Bordes et al. 2013)	263	251	75.4	89.2	243	125	34.9	47.1
TransH (unif) (Wang et al. 2014)	318	303	75.4	86.7	211	84	42.5	58.5
TransH (bern) (Wang et al. 2014)	401	388	73.0	82.3	212	87	45.7	64.4
TransR (unif)	232	219	78.3	91.7	226	78	43.8	65.5
TransR (bern)	238	225	**79.8**	92.0	**198**	77	48.2	68.7
CTransR (unif)	243	230	78.9	**92.3**	233	82	44	66.3
CTransR (bern)	**231**	**218**	79.4	**92.3**	199	**75**	**48.4**	**70.2**

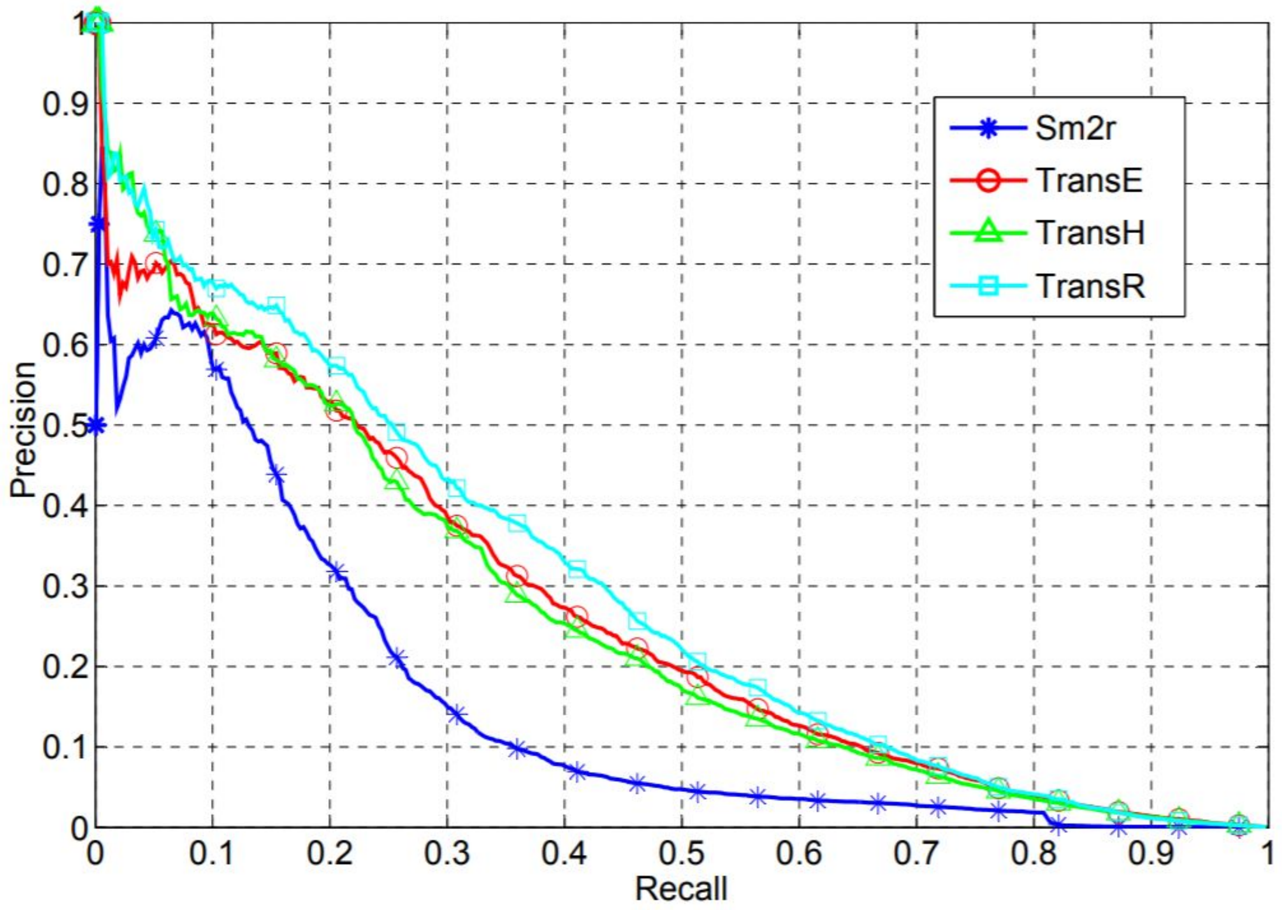
EXPERIMENT RESULTS

Triple Classification: judging whether a given triple (*h*, *r*, *t*) is correct

Data Sets	WN11	FB13	FB15K
SE	53.0	75.2	-
SME (bilinear)	70.0	63.7	-
SLM	69.9	85.3	-
LFM	73.8	84.3	-
NTN	70.4	**87.1**	68.5
TransE (unif)	75.9	70.9	79.6
TransE (bern)	75.9	81.5	79.2
TransH (unif)	77.7	76.5	79.0
TransH (bern)	78.8	83.3	80.2
TransR (unif)	85.5	74.7	81.7
TransR (bern)	**85.9**	82.5	83.9
CTransR (bern)	85.7	-	**84.5**

EXPERIMENT RESULTS

Relation Extraction from Text: Combining results from text-based relation extraction model and knowledge graph embeddings to rank test triples



MY THOUGHTS

- Training time – Performance Tradeoff
- A single CNN instead of matrix for each relation
- Relation hyperplane vs. relation space
- CTransR is more inspirational

Gated Graph Sequence Neural Networks

Li, Y., Tarlow, D., Brockschmidt, M., & Zemel, R, ICLR 2016
Presented by Hyoungwook Nam (hn5)

Abstract

- **Graph-structured data** appears on many domains

- Based on **GNNs** (graph neural network), utilize **GRU** (gated recurrent unit) and extend to output **sequences**

- The result is **flexible**, and better than sequence-based models (e.g. LSTM) if a problem can be **graph-structured**

- State-of-the-art on **bAbI** and **graph algorithm** tasks

Introduction

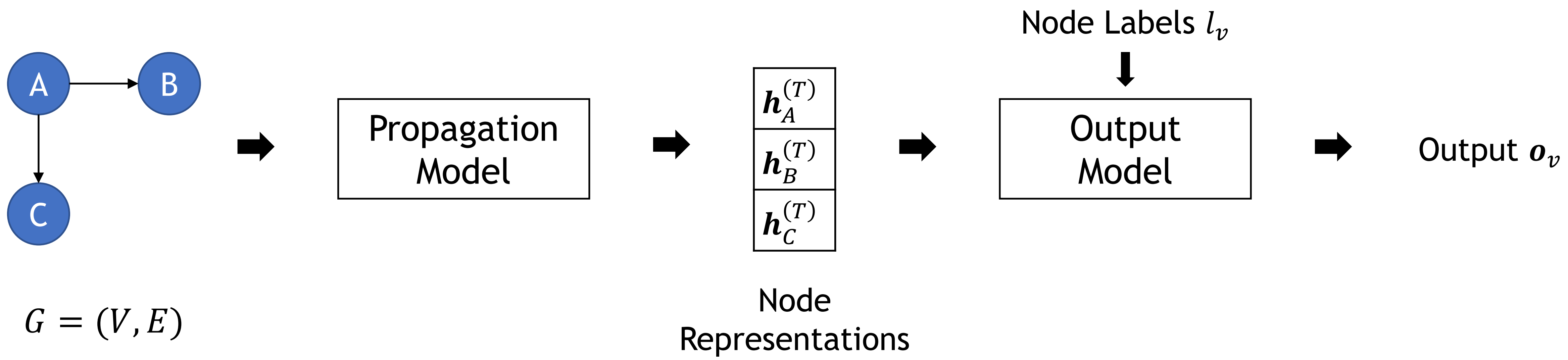
Previous approaches:

- Graph feature engineering, Graph neural network (GNN), spectral networks, etc.

Contributions:

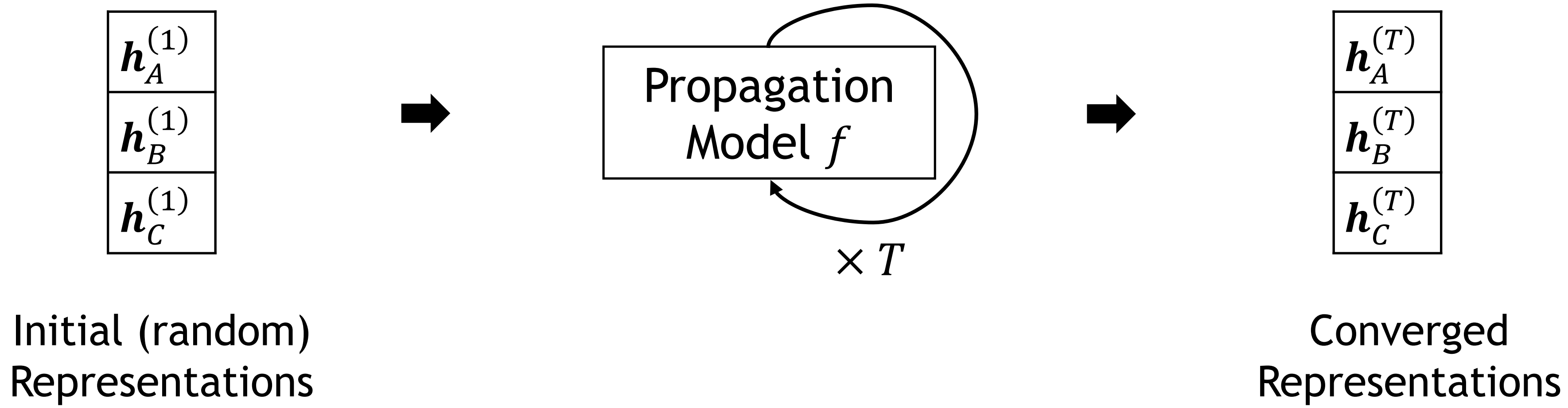
- Propose GGS-NN, a **gated** GNN for **sequence** output.
- Show that it is useful for many problems (shortest path, program verification, etc.)

Graph Neural Network (GNN)



- Propagation model gives node representations (embeddings)
- **Output model** g provides outputs $o_v = g(\boldsymbol{h}_v, l_v)$ per vertex
- Similar to RNN encoder-decoder without attention

Propagation Model



- $\boldsymbol{h}_v^{(t)} = f(NBR_v^{(t-1)})$ where NBR_v is a set of v’s **neighbors**
- From initial $\boldsymbol{h}_v^{(1)}$s, the update **repeats** until **convergence**

Gated Graph Neural Network (GG-NN)

- Initialize $h_v^{(1)}$ with **annotations** x_v instead of random values
- **GRU-like** propagation model

$$\mathbf{h}_v^{(1)} = [\boldsymbol{x}_v^\top, \mathbf{0}]^\top \quad (1)$$

$$\mathbf{a}_v^{(t)} = \mathbf{A}_{v:}^\top \left[\mathbf{h}_1^{(t-1)\top} \dots \mathbf{h}_{|\mathcal{V}|}^{(t-1)\top}\right]^\top + \mathbf{b} \quad (2)$$

$$\mathbf{z}_v^t = \sigma\left(\mathbf{W}^z \mathbf{a}_v^{(t)} + \mathbf{U}^z \mathbf{h}_v^{(t-1)}\right) \quad (3)$$

$$\mathbf{r}_v^t = \sigma\left(\mathbf{W}^r \mathbf{a}_v^{(t)} + \mathbf{U}^r \mathbf{h}_v^{(t-1)}\right) \quad (4)$$

$$\widetilde{\mathbf{h}_v^{(t)}} = \tanh\left(\mathbf{W}\mathbf{a}_v^{(t)} + \mathbf{U}\left(\mathbf{r}_v^t \odot \mathbf{h}_v^{(t-1)}\right)\right) \quad (5)$$

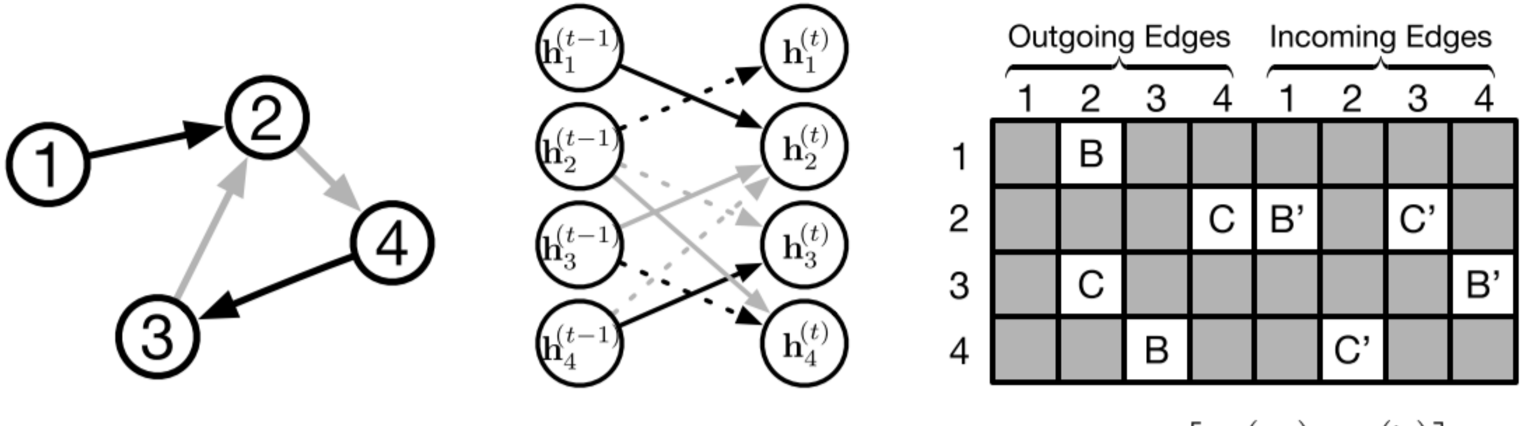
$$\mathbf{h}_v^{(t)} = (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{(t-1)} + \mathbf{z}_v^t \odot \widetilde{\mathbf{h}_v^{(t)}}. \quad (6)$$

- Output model: Graph-level or node-selection with softmax

$$\mathbf{h}_{\mathcal{G}} = \tanh\left(\sum_{v \in \mathcal{V}} \sigma\left(i(\mathbf{h}_v^{(T)}, \boldsymbol{x}_v)\right) \odot \tanh\left(j(\mathbf{h}_v^{(T)}, \boldsymbol{x}_v)\right)\right)$$

$$o_v = g(\mathbf{h}_v^{(T)}, \boldsymbol{x}_v)$$

Adjacency Matrix and Neighborhood

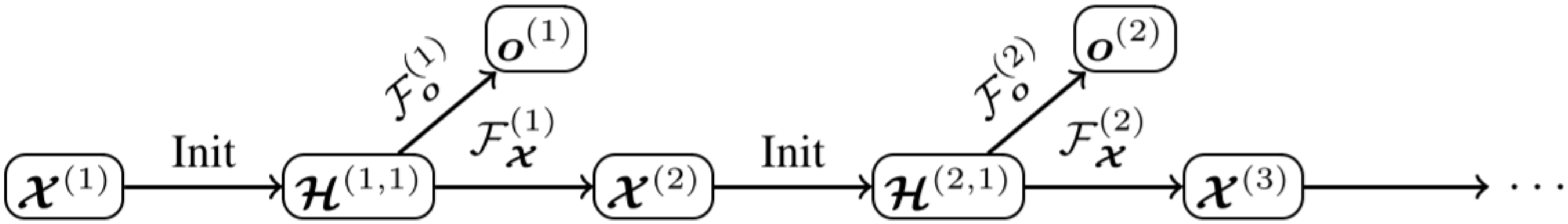


(a) (b) (c) $\mathbf{A} = \left[\mathbf{A}^{(\mathrm{out})}, \mathbf{A}^{(\mathrm{in})}\right]$

- Adjacency matrix $A = [A^{(out)}, A^{(in)}]$ for neighborhood updates
- $a_v^{(t)} = A^T \left[h_1^{(t-1)} \ldots h_V^{(t-1)}\right]^T$ will propagate $h_{v'}$ of v's neighbors
- $h_v^{(t)} = GRU(a_v^{(t)}, h_v^{(t-1)})$

Gated Graph Sequence NN (GGS-NN)

- Objective: create an output sequence $o^{(1)} \dots o^{(k)}$
- RNN-like structure using two GG-NNs $F_o^{(n)}, F_k^{(n)}$

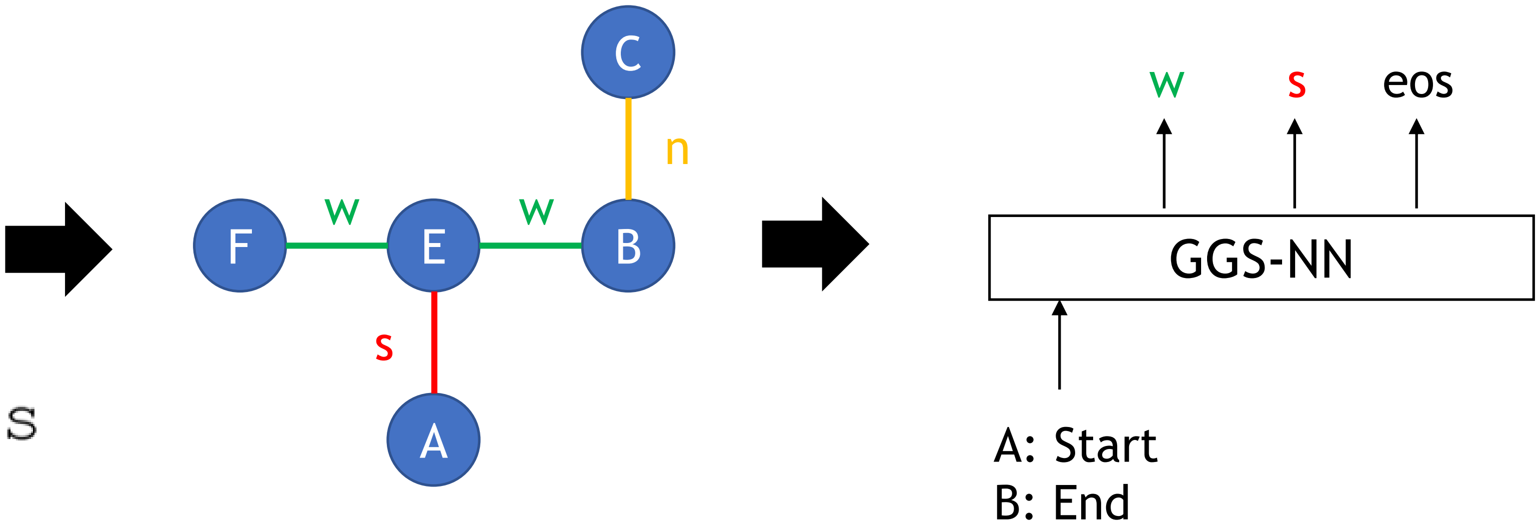


- Latent (hidden) or observed annotations $X^{(n)}$s are possible

bAbI Task Evaluation Setup

- Symbolic task to graph structured problem

```
E s A
B n C
E w F
B w E
eval path B A w,s
```



bAbI + Graph Algorithm Result

- (N): Samples needed for the best result (max 950)

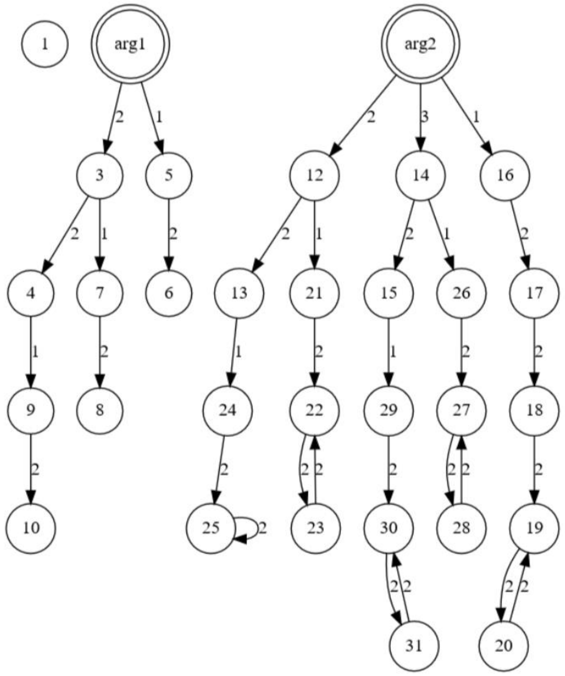
Task	RNN	LSTM	GG-NN
bAbI Task 4	97.3±1.9 (250)	97.4±2.0 (250)	100.0±0.0 (50)
bAbI Task 15	48.6±1.9 (950)	50.3±1.3 (950)	100.0±0.0 (50)
bAbI Task 16	33.0±1.9 (950)	37.5±0.9 (950)	100.0±0.0 (50)
bAbI Task 18	88.9±0.9 (950)	88.9±0.8 (950)	100.0±0.0 (50)

Task	RNN	LSTM	GGS-NNs		
bAbI Task 19	24.7±2.7 (950)	28.2±1.3 (950)	71.1±14.7 (50)	92.5±5.9 (100)	99.0±1.1 (250)
Shortest Path	9.7±1.7 (950)	10.5±1.2 (950)	100.0± 0.0 (50)		
Eulerian Circuit	0.3±0.2 (950)	0.1±0.2 (950)	100.0± 0.0 (50)		

Program Verification Setup

- Program → Memory Heap → GG-NN → Invariant Logic

```
node* concat(node* a, node* b) {
  if (a == NULL) return b;
  node* cur = a;
  while (cur.next != NULL)
    cur = cur->next;
  cur->next = b;
  return a;                     }
```



GGS-NN

$$\mathsf{ls}(\mathtt{arg1}, \mathtt{NULL}, \lambda t_1 \to \mathsf{ls}(t_1, \mathtt{NULL}, \top)) * \mathsf{tree}(\mathtt{arg2}, \lambda t_2 \to \exists e_1.\mathsf{ls}(t_2, e_1, \top) * \mathsf{ls}(e_1, e_1, \top))$$

Program Verification Result

- Exceeds the previous method with domain-specific feature engineering (89.96% > 89.11%)

Program	Invariant Found
Traverse1	$\mathsf{ls}(\mathtt{lst}, \mathtt{curr}) * \mathsf{ls}(\mathtt{curr}, \mathtt{NULL})$
Traverse2	$\mathtt{curr} \neq \mathtt{NULL} * \mathtt{lst} \neq \mathtt{NULL} * \mathsf{ls}(\mathtt{lst}, \mathtt{curr}) * \mathsf{ls}(\mathtt{curr}, \mathtt{NULL})$
Concat	$\mathtt{a} \neq \mathtt{NULL} * a \neq b * b \neq \mathtt{curr} * \mathtt{curr} \neq \mathtt{NULL}$ $* \mathsf{ls}(\mathtt{curr}, \mathtt{NULL}) * \mathsf{ls}(a, \mathtt{curr}) * \mathsf{ls}(b, \mathtt{NULL})$
Copy	$\mathsf{ls}(\mathtt{curr}, \mathtt{NULL}) * \mathsf{ls}(\mathtt{lst}, \mathtt{curr}) * \mathsf{ls}(\mathtt{cp}, \mathtt{NULL})$
Dispose	$\mathsf{ls}(\mathtt{lst}, \mathtt{NULL})$
Insert	$\mathtt{curr} \neq \mathtt{NULL} * \mathtt{curr} \neq \mathtt{elt} * \mathtt{elt} \neq \mathtt{NULL} * \mathtt{elt} \neq \mathtt{lst} * \mathtt{lst} \neq \mathtt{NULL}$ $* \mathsf{ls}(\mathtt{elt}, \mathtt{NULL}) * \mathsf{ls}(\mathtt{lst}, \mathtt{curr}) * \mathsf{ls}(\mathtt{curr}, \mathtt{NULL})$
Remove	$\mathtt{curr} \neq \mathtt{NULL} * \mathtt{lst} \neq \mathtt{NULL} * \mathsf{ls}(\mathtt{lst}, \mathtt{curr}) * \mathsf{ls}(\mathtt{curr}, \mathtt{NULL})$

Takeaways

- GNNs consist of a **propagation model** to update node representations and an **output model** to compute the outputs

- **GG-NN** uses a **GRU-like** propagation model and **GGS-NN** follows the recurrent structure for **sequential outputs**

- They are proven very powerful on tasks like bAbI and program verification which can be **graph-structured**

Graph CNNs for Semantic Role Labeling

Eddie Huang

`marcheggiani-titov-2017-encoding`
"Encoding Sentences with Graph Convolutional Networks for Semantic Role Labeling" - Marcheggiani, Diego and Titov, Ivan

20 February, 2020

Outline

Outline

Main Idea

Main Idea

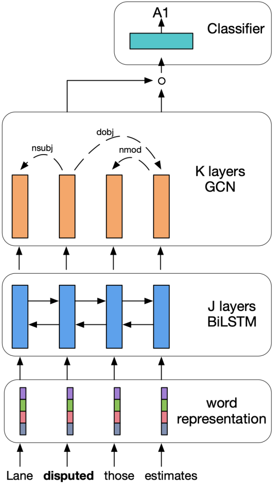


Figure 1: Model Architecture

Main Idea

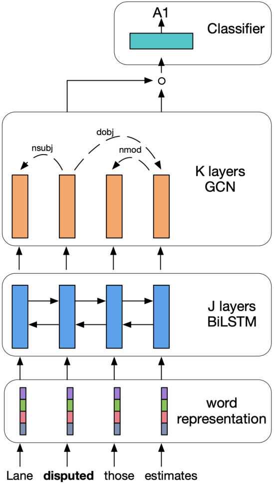


Figure 1: Model Architecture

A new model using graph convolutional neural networks with syntax graphs exceeds previous best models in semantic role labeling

Outline

What is Semantic Role Labeling (SRL)?

What is Semantic Role Labeling (SRL)?

Want to know **"who did what to whom?"**

What is Semantic Role Labeling (SRL)?

Want to know **"who did what to whom?"**

Example

`Sequa makes and repairs jet engines`

What is Semantic Role Labeling (SRL)?

Want to know **"who did what to whom?"**

Example

`Sequa makes and repairs jet engines`

- Predicates: `makes`, `repairs`
- Semantic Roles:
  - Agent: `Sequa`
  - Patient: `engines`

Why do we want SRL?

Why do we want SRL?

Figure 2: SRL provides more intermediate features in NLP pipeline

Related Work

Related Work

- Earliest works with RNNs on SRL began in 2008
- 2014-2017 Modern approaches using LSTMs and Syntactic features
- A multi-layer Bi-LSTM model made in 2017 was the most state-of-the-art SRL model at the time (created by the same author)

Basic Components

- GCNs
- Syntax Parsing
- LSTMs
- Word Embeddings

Outline

Reiterate Main Idea

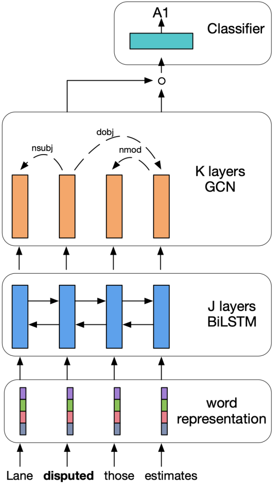


Figure 3: Model Architecture

A new model using graph convolutional neural networks with syntax graphs exceeds previous best models in semantic role labeling

Example

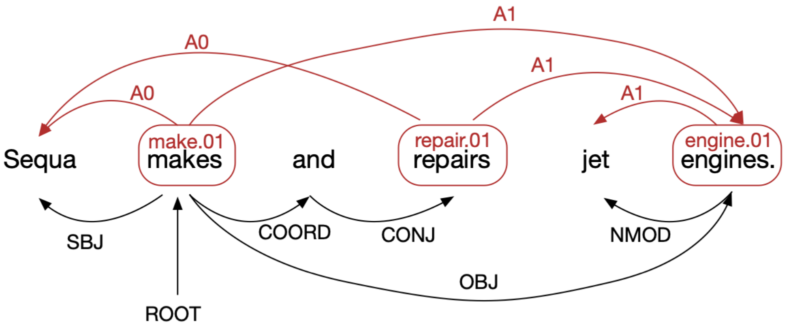


Figure 4: An Example (red is what we want to find)

Outline

Syntactic Dependency Graph

- Syntax of a language can be represented as a relationship between words rooted at the predicate of a sentence

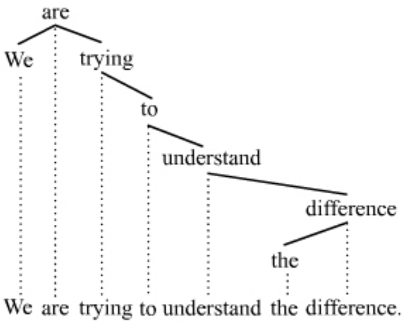


Figure 5: A syntax dependency graph

Syntactic Dependency Graph

- Syntax of a language can be represented as a relationship between words rooted at the predicate of a sentence
- Edges represent the syntactic relationship between the nodes

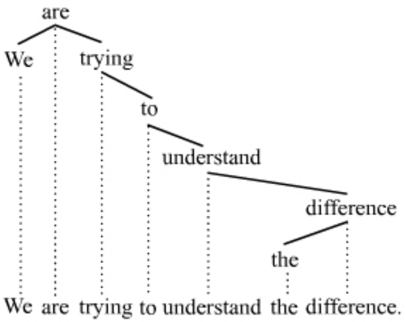


Figure 5: A syntax dependency graph

Role of Syntactic Dependency Graphs

Role of Syntactic Dependency Graphs

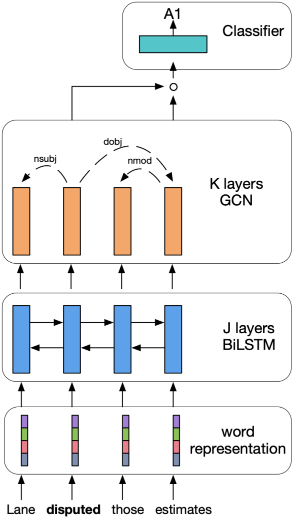


Figure 6: Syntactic dependency occurs between LSTM and GCN

What are Graph Convolutional Neural Networks (GCNs)?

What are Graph Convolutional Neural Networks (GCNs)?

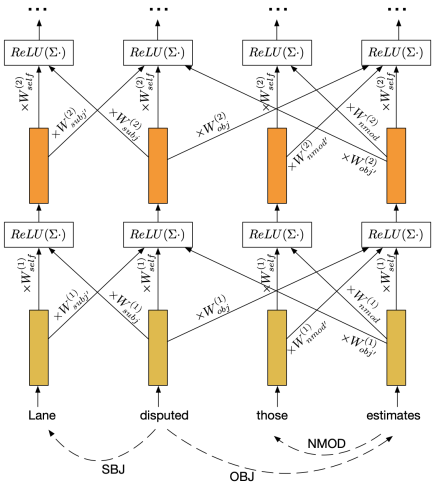


Figure 7: Graph Convolutional Neural Network

What are Graph Convolutional Neural Networks (GCNs)?

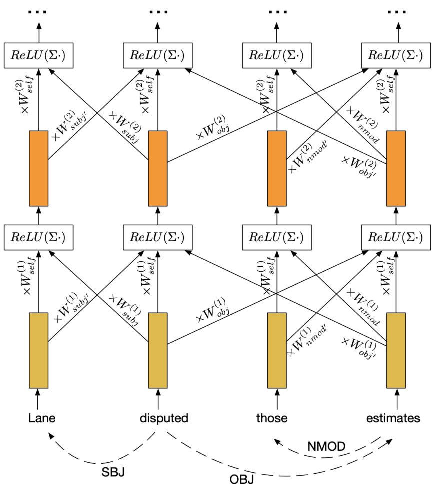


Figure 7: Graph Convolutional Neural Network

GCNs are neural networks that take in a graph (a set of nodes and edges) and output features for each node.

How do GCNs compute features for nodes?

How do GCNs compute features for nodes?

Node features are computed as non-linear combinations of their neighbors

How do GCNs compute features for nodes?

Node features are computed as non-linear combinations of their neighbors

$$h_v = ReLU\Bigg(\sum_{u \in N_(v)} (Wx_u + b)\Bigg)$$

- x_u is a vector representation of node u.

How do GCNs compute features for nodes?

Node features are computed as non-linear combinations of their neighbors

$$h_v = ReLU\Big(\sum_{u \in N_(v)} (Wx_u + b)\Big)$$

- x_u is a vector representation of node u.

Can stack k GCN layers to capture dependency between nodes k hops away ($k = 1$ was best)

$$h_v^{(k)} = ReLU\Big(\sum_{u \in N_(v)} (W^{(k-1)} h_u^{(k-1)} + b^{(k-1)})\Big)$$

K Layers Captures K-hop dependencies

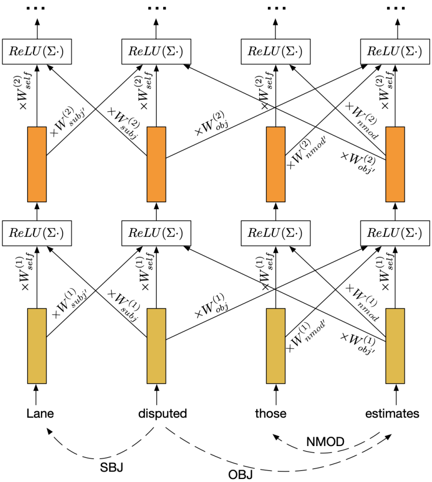


Figure 8: $h_v^{(k)} = ReLU\left(\sum_{u \in N_(v)} (W^{(k-1)} h_u^{(k-1)} + b^{(k-1)}) \right)$

Capturing Edge Information

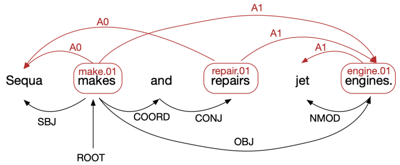


Figure 9: Syntax graphs have directionality and edges have different meanings based on their syntax

Capturing Edge Information

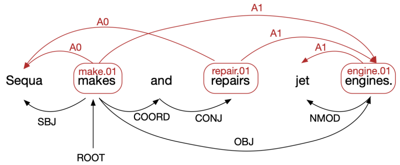


Figure 9: Syntax graphs have directionality and edges have different meanings based on their syntax

Solution - Have separate weights for each type of edge

Capturing Edge Information

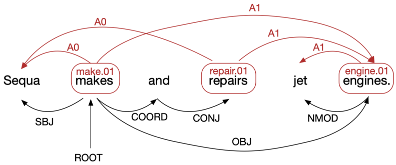


Figure 9: Syntax graphs have directionality and edges have different meanings based on their syntax

Solution - Have separate weights for each type of edge

$$h_v^{(k)} = ReLU\Bigg(\sum_{u \in N_(v)} (W_{dir(\mathbf{u},\mathbf{v})}^{(k-1)} h_u^{(k-1)} + b_{\mathbf{L}(\mathbf{u},\mathbf{v})}^{(k-1)}) \Bigg)$$

- $dir(u, v) \in \{backward(1), \textit{self-loop}(2), forward(3)\}$
- $L(u, v)$ captures both directionality and syntax function

Weighting Importance to Different Syntax

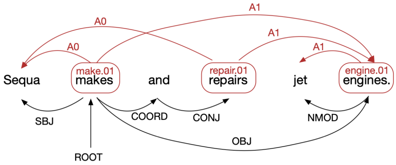


Figure 10: Some edges are more important than others

Weighting Importance to Different Syntax

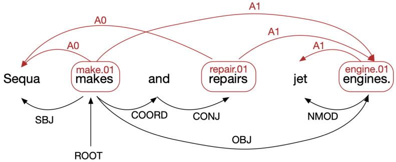


Figure 10: Some edges are more important than others

Solution - Use sigmoid to express weighted importance

Weighting Importance to Different Syntax

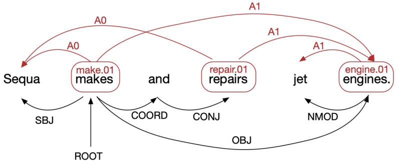


Figure 10: Some edges are more important than others

Solution - Use sigmoid to express weighted importance

$$g_{u,v}^{(k-1)} = \sigma\big(h_u^{(k-1)} \cdot \hat{v}_{dir(u,v)}^{(k-1)} + \hat{b}_{L(u,v)}^{(k-1)}\big)$$

Weighting Importance to Different Syntax

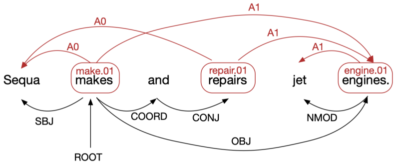


Figure 10: Some edges are more important than others

Solution - Use sigmoid to express weighted importance

$$g_{u,v}^{(k-1)} = \sigma(h_u^{(k-1)} \cdot \hat{v}_{dir(u,v)}^{(k-1)} + \hat{b}_{L(u,v)}^{(k-1)})$$

$$h_v^{(k)} = ReLU\left(\sum_{u \in N_(v)} \mathbf{g_{u,v}^{(k-1)}} \left(W_{dir(u,v)}^{(k-1)} h_u^{(k-1)} + b_{L(u,v)}^{(k-1)} \right) \right)$$

Final Version of GCN

Node features are computed as a weighted non-linear combination of neighbors within k hops.

$$h_v^{(k)} = ReLU\left(\sum_{u \in N_(v)} g_{u,v}^{(k-1)} \left(W_{dir(u,v)}^{(k-1)} h_u^{(k-1)} + b_{L(u,v)}^{(k-1)} \right) \right)$$

Remark

Similar to a multi-layer perceptron

Architecture

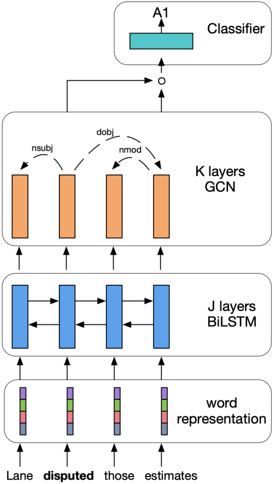


Figure 11: Architecture of new model

Remarks

- Relies on external syntactic parser and predicate identifier.
- Layer after the GCN is just another feed-forward network with a softmax for semantic role classification.

LSTMs and GCNs compliment each other

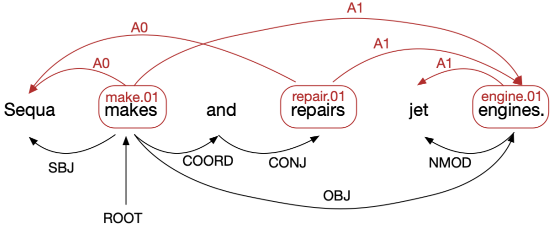


Figure 12: **engines** is physically far away from **makes** but syntactically adjacent to it

LSTMs (RNNs) efficiently capture physically close dependencies.
GCNs can efficiently capture physically far away dependencies

Outline

Results

Precision, recall, and F1 scores for the CoNLL-2009 English and Chinese datasets

System	P	R	F_1
Lei et al. (2015) (local)	-	-	86.6
FitzGerald et al. (2015) (local)	-	-	86.7
Roth and Lapata (2016) (local)	88.1	85.3	86.7
Marcheggiani et al. (2017) (local)	88.7	86.8	87.7
Ours (local)	**89.1**	**86.8**	**88.0**
Björkelund et al. (2010) (global)	88.6	85.2	86.9
FitzGerald et al. (2015) (global)	-	-	87.3
Foland and Martin (2015) (global)	-	-	86.0
Swayamdipta et al. (2016) (global)	-	-	85.0
Roth and Lapata (2016) (global)	90.0	85.5	87.7
FitzGerald et al. (2015) (ensemble)	-	-	87.7
Roth and Lapata (2016) (ensemble)	90.3	85.7	87.9
Ours (ensemble 3x)	**90.5**	**87.7**	**89.1**

Figure 13: English Results

System	P	R	F_1
Zhao et al. (2009) (global)	80.4	75.2	77.7
Björkelund et al. (2009) (global)	82.4	75.1	78.6
Roth and Lapata (2016) (global)	83.2	75.9	79.4
Ours (local)	**84.6**	**80.4**	**82.5**

Figure 14: Chinese Results

Remark

- Beats previous best results by 0.6% – 1.9%
- $k = 1$ works best

Outline

Criticism

Syntactic graph parsing is similar to semantic role labeling because their graph structures look nearly the same. Could probably make at least a decent hand-made algorithm to perform SRL given syntax dependency graph. Would like to see comparison between hand-made algorithm vs. neural net.

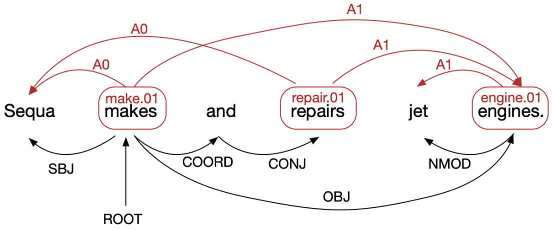


Figure 15: SRL and Syntactic are nearly identical