Carnegie Mellon University

Semi-supervised Classification With Graph Convolutional Networks (2017)

Thomas N. Kipf, Max Welling

Presenter: Hui Chen, Machine Learning Department

Outline

1. What is a graph neural network (GNN)?
2. Problem setting of the paper
3. Previous methods
4. Formulation of graph convolutional network(GCN)
5. Theory behind GCN
6. Questions and my answers

What is a GNN?

GNNs are NNs that operate on graph-structured data.

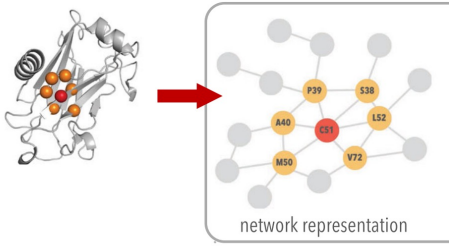
Why do we need GNNs?

1. To capture correlation inside sample.
2. To capture correlation across samples.

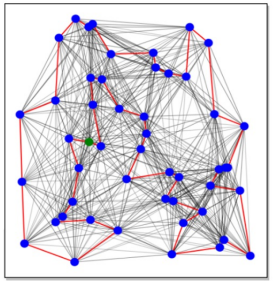
Examples:

GNN example

1. To capture correlation inside sample



Protein structure [1]

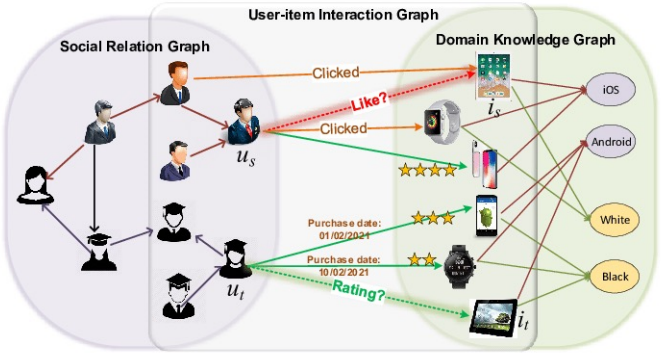


Travelling salesman problem [2]

[1] Vladimir Gligorijević. et al. Structure-based protein function prediction using graph convolutional networks. 2021
[2] Chaitanya K. Joshi. et al. An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem. 2019

GNN example

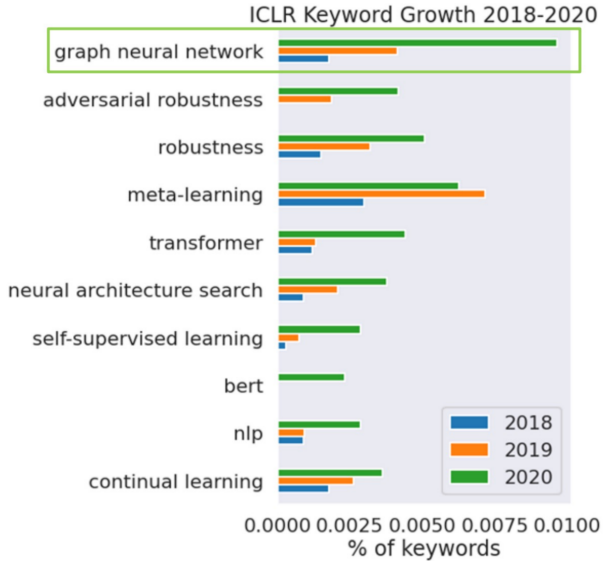
2. To capture correlation across sample



Recommender system [1]
E.g., Pinterest [2]

[1] Shoujin Wang. et al. Graph Learning based Recommender Systems: A Review. 2021
[2] Rex Ying. et al. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. 2018

Graph Neural Networks: The Hottest Subfield in ML



Taken from slides of Prof. Jia

Carnegie Mellon University

Problem setting

Undirected graph $\mathrm{G} = (V, E)$, where $|V| = N$.
Node $v_i \in V$ has a feature vector $X_i \in R^d$.
Edge $(v_i, v_j) \in E$ can be weighted or unweighted.
Some nodes are labeled, and the task is to make predictions to those unlabeled.

Previous methods

Vanilla neural network + graph information for regularization:

Main idea: penalize $\| f(x_i) - f(x_j) \|^2$ for each edge $(v_i, v_j) \in E$, where f is a neural network.

Formulation: e.g., $reg = \sum_{i,j} A_{ij} \| f(x_i) - f(x_j) \|^2$, where A is the adjacency matrix (binary or weighted).

Deficiency: assume that connected nodes tend to share the same label, which might be violated in practice.

Graph convolutional network(GCN)

GCN Layer:

$$\sigma(\hat{A}XW)$$

σ: activation function

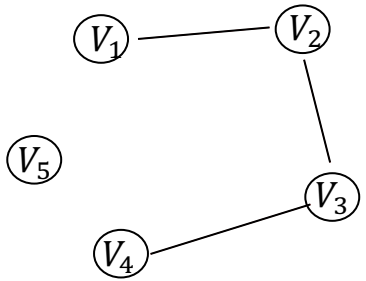
X: input matrix $\in R^{N*d}$

W: parameter matrix $\in R^{d*d'}$

Graph Laplacian $\hat{A} = \widetilde{D}^{-1/2}\tilde{A}\widetilde{D}^{-1/2}$, where $\tilde{A} = A + I$, $\widetilde{D}$ is a diagonal matrix with $\widetilde{D}_{ii} = \sum_j \tilde{A}_{ij}$.

Example of GCN layer

$\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$, where $\tilde{A} = A + I$, $\tilde{D}$ is a diagonal matrix with $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$.



$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\tilde{A} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\tilde{D}^{-1/2} = \begin{bmatrix} 0.71 & 0. & 0. & 0. & 0. \\ 0. & 0.58 & 0. & 0. & 0. \\ 0. & 0. & 0.58 & 0. & 0. \\ 0. & 0. & 0. & 0.71 & 0. \\ 0. & 0. & 0. & 0. & 1. \end{bmatrix}$$

$$\hat{A} = \begin{bmatrix} 0.5 & 0.41 & 0. & 0. & 0. \\ 0.41 & 0.33 & 0.33 & 0. & 0. \\ 0. & 0.33 & 0.33 & 0.41 & 0. \\ 0. & 0. & 0.41 & 0.5 & 0. \\ 0. & 0. & 0. & 0. & 1. \end{bmatrix}$$

Carnegie Mellon University

Example of GCN layer

$\hat{A} =$

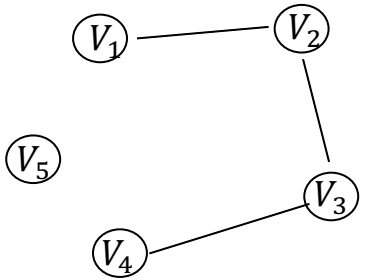
```
[[0.5  0.41 0.   0.   0.  ]
 [0.41 0.33 0.33 0.   0.  ]
 [0.   0.33 0.33 0.41 0.  ]
 [0.   0.   0.41 0.5  0.  ]
 [0.   0.   0.   0.   1.  ]]
```

Except for the diagonals, $\hat{A}$ has the same pattern of non-zero entries with A

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \Rightarrow \hat{A}X = \begin{bmatrix} 0.5x_1 + 0.41x_2 \\ 0.41x_1 + 0.33x_2 + 0.33x_3 \\ 0.33x_2 + 0.33x_3 + 0.41x_4 \\ 0.41x_4 + 0.5x_5 \\ x_5 \end{bmatrix}$$

Example of GCN layer

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \Rightarrow \hat{A}X = \begin{bmatrix} 0.5x_1 + 0.41x_2 \\ 0.41x_1 + 0.33x_2 + 0.33x_3 \\ 0.33x_2 + 0.33x_3 + 0.41x_4 \\ 0.41x_4 + 0.5x_5 \\ x_5 \end{bmatrix}$$



Convolution is just weighted sum of a node's feature and its neighbors' features.
aka *message passing and aggregation*

GCN layer: $\sigma(\hat{A}XW)$
W: feature transformation
$\hat{A}$: message passing and aggregation

Why GCNs work?

In essence, GCN layer is an approximated spectral convolution.

Consider a signal $x \in R^N$ (each node has a scalar), a filter $g_\theta = \text{diag}(\theta)$ parameterized by $\theta \in R^N$ in the Fourier domain, i.e.

$$
\begin{aligned}
g_\theta \star x &\overset{(1)}{=} U g_\theta U^\top x \\
&\overset{(2)}{\approx} \sum_{k=0}^{K} \theta_k T_k\left(\tilde{L}\right) x \\
&\overset{(3)}{\approx} \theta_0 x - \theta_1 D^{-1/2} A D^{-1/2} x \\
&\overset{(4)}{\approx} \theta\left(I_N + D^{-1/2} A D^{-1/2}\right) x \\
&\overset{(5)}{\approx} \theta \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} x,
\end{aligned}
$$

where (1) U is the eigenvectors of $L = I_N - D^{-1/2} A D^{-1/2}$; (2) uses K-th order Chebyshev polynomials, $\tilde{L} = \frac{2}{\lambda_{\max}} L - I_N$; (3) sets $K = 1$; (4) assumes $\theta_0 + \theta_1 = 0$; (5) uses the renormalization trick $\tilde{A} = A + I_N$.

The form $\hat{A}XW$ is the generalization of the formula above.

Carnegie Mellon University

Why GCNs work?

It is not a new idea to approximate the spectral convolution, such as [1].

What's special about GCN?

Two main streams of GNN architectures:

1. Spectral-based.

spectral graph theory, eigendecomposition, matrix multiplication, sound theory but inefficient implementation

2. Spatial-based.

Message passing among nodes, lack of theory but efficient implementation

GCN is at the intersection of these two main streams!

[1] Michael Defferrard. et al. Convolutional neural networks on graphs with fast localized spectral filtering. 2016

GCN structure in the paper

$$Z = f(X, A) = \mathrm{softmax}\Big(\hat{A}\ \mathrm{ReLU}\Big(\hat{A}XW^{(0)}\Big)\ W^{(1)}\Big)$$

--- Why only two layers?

--- Because deep GCNs do not perform well. Why?

--- An intuitive explanation is, graph convolution can be viewed as information exchange between neighbors, and if we keep doing this, all nodes' features will become more and more similar.

GCN structure in the paper

$$Z = f(X, A) = \mathrm{softmax}\Big(\hat{A}\ \mathrm{ReLU}\Big(\hat{A}XW^{(0)}\Big)\ W^{(1)}\Big)$$

--- Graph Laplacian $\hat{A}$ has a smoothing effect. [1] proves that if we apply the graph Laplacian enough times, all nodes' features will converge to the same value. Hence the name *over-smoothing*.

-- Still an open question. Researchers try to explain it from various perspectives, such as Markov Process [2], Neural Tangent Kernel [3].

[1] Qimai Li. et al. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. 2018
[2] Kenta Oono. et al. GRAPH NEURAL NETWORKS EXPONENTIALLY LOSE EXPRESSIVE POWER FOR NODE CLASSIFICATION, 2020
[3] Wei Huang. et al. Towards Deepening Graph Neural Networks: A GNTK-based Optimization Perspective. 2022

GCN structure in the paper

$$Z = f(X, A) = \mathrm{softmax}\Big(\hat{A}\ \mathrm{ReLU}\Big(\hat{A}XW^{(0)}\Big)\ W^{(1)}\Big)$$

--- What if we need deep GCNs?

--- Just stack two GCN layers after a deep neural network.

Experiment results from the paper

Table 2: Summary of results in terms of classification accuracy (in percent).

Method	**Citeseer**	**Cora**	**Pubmed**	**NELL**
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [28]	59.6	59.0	71.1	26.7
LP [32]	45.3	68.0	63.0	26.5
DeepWalk [22]	43.2	67.2	65.3	58.1
ICA [18]	69.1	75.1	73.9	23.1
Planetoid* [29]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)
GCN (rand. splits)	67.9 ± 0.5	80.1 ± 0.5	78.9 ± 0.7	58.4 ± 1.7

Carnegie Mellon University

Summary

GCN Layer:

$$\sigma(\hat{A}XW)$$

Approximation to spectral convolution.
Intersection of spectral-based and spatial-based GNNs.
Two layers are enough.

Questions

1. After reading the paper, someone claims*: graph convolution is matrix multiplication, which takes $O(n^3)$, so it won't work for large-scale problems*. Do you agree?
2. Is there any connection between graph neural networks and the self-attention mechanism?
3. GCN still relies on the assumption that connected nodes tend to be in the same class, though not as heavily as previous methods. What if we want to apply GCN when this assumption is violated?

Answer Q1

torch_geometric: sparse matrix multiplication

```
from torch_sparse import matmul
def message_and_aggregate(self, adj_t, x):
        return matmul(adj_t, x, reduce=self.aggr)
```

dgl: message passing

```
graph.apply_edges(lambda e: {'_norm_edge_weights': \
    e.src['_src_out_w'] * e.dst['_dst_in_w'] * e.data['_edge_w']})
```

If the graph is large, a common practice is not to load the whole graph into the memory. Instead, we sample some subgraph in each iteration.

Answer Q2

On one hand, the blog post [1] claims that transformers are, in essence, GNNs.

On the other hand, [2] proposes Graphormer, an extension of Transformer, and claims that with specific choice of hyper-parameters, Graphormer can represent popular GNN models including GCN.

A recent work [3] (ICLR 2022) tries to analyze over-smoothing problem of BERT from a GCN perspective.

[1] https://towardsdatascience.com/transformers-are-graph-neural-networks-bca9f75412aa
[2] Chengxuan Ying. et al. Do Transformers Really Perform Bad for Graph Representation? 2021
[3] Han Shi. et al. Revisiting Over-smoothing in BERT from the Perspective of Graph. 2022

Thank you for listening!

Inductive Representation Learning on Large Graphs

William L. Hamilton, Rex Ying, Jure Leskovec

Paper presentation prepared by

Paola A. Buitrago

March 21st, 2022

15-849: Machine Learning Systems

Carnegie Mellon University, Pittsburgh, PA

Overview

- Context: Large data graphs used for prediction and graph analysis tasks.
- Low-dimensional vector embeddings:
  - are a dense vector where high-dimensional information about a node graph's neighborhood.
  - are produced using dimensionality reduction techniques.
  - in large graphs have proved extremely useful as feature inputs for a wide variety of prediction and graph analysis tasks.
- SOTA approach
  - Generation of node vector embeddings from a single fixed graph.

Overview

- Solution
  - For production ML systems, they operate on evolving graphs and constantly encounter unseen nodes.
  - Real world applications require embeddings to be quickly generated for unseen nodes or entirely new (sub) graphs.
  - Inductive approach to generating node embeddings.
  - Generation of low dimensional vector embeddings for unseen nodes or entirely new (sub) graphs.

Challenges

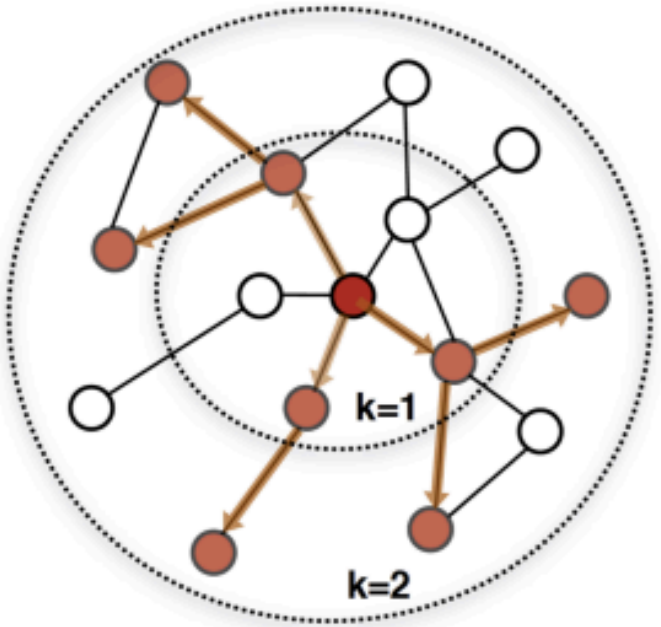
- Considering unseen nodes requires “Aligning” newly observed subgraphs to the node embeddings that the algorithm has already optimized on.
- “Algorithm needs to learn to recognize structural properties of a node’s neighborhood that reveal both the node’s local role in the graph, as well as its global position”
- Need a solution that can be implemented in a computationally reasonable time.

Proposed Approach

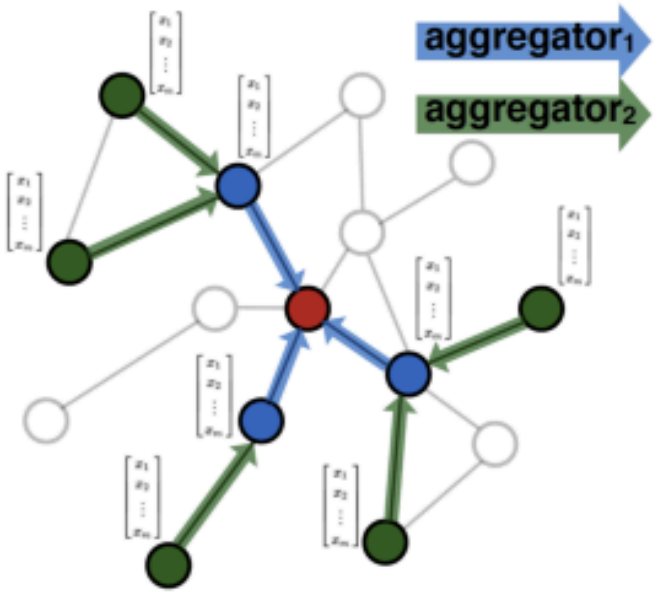
GraphSAGE (SAmple and aggreGatE)

- Approach that leverages node features in order to learn an embedding function that generalizes to unseen nodes.
- It can also make use of graph structural features and applied to graph without node features.
- Train a set of aggregator functions that learn to aggregate feature information from a node's local neighborhood.
- At test or inference time, generate embeddings for entirely unseen nodes by applying the learned aggregate functions.
- Can be trained without task-specific supervision, and also on a fully supervised manner.

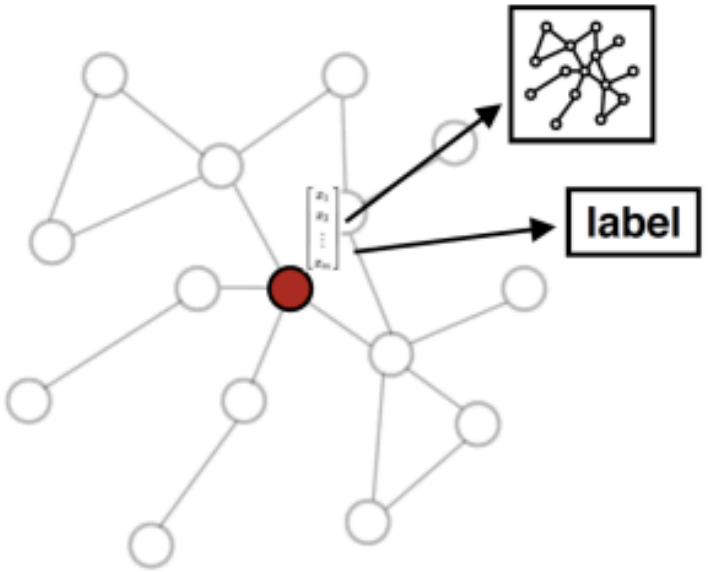
GraphSAGE (Graph **SA**mple and aggre**G**at**E**)



1. Sample neighborhood



2. Aggregate feature information from neighbors



3. Predict graph context and label using aggregated information

GraphSAGE - Embedding Generation Algorithm

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 $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

GraphSAGE – Neiborhood Definition

To keep computational footprint fixed, and memory and expected runtime predictable:

- Uniformly sample a fixed-size set of neighbors.
- Draw different uniform samples at each iteration k in the algorithm 1.

GraphSAGE – Learning the Parameters

Unsupervised approach

- Apply graph-based loss function to the output representations.
- Tune weight matrices and parameters of aggregator functions via SGD.

Supervised approach

- Representations to be used on a specific downstream task
- Replace or augment unsupervised loss with a task specific objective (e.g. cross-entropy loss)

GraphSAGE - Aggregators

Agregator	Description	Advantages	Limitations
Mean aggregator	Elementwise mean on vectors of neighbor set		
LSTM aggregator	LSTM architecture		
Pooling aggregator	Vector fed to a FC NN + elementwise max pooling operation		

Evaluation

- Test ability to generate useful embeddings on unseen data:
- Evaluated on three node-classification benchmarks.

1. Inductive Learning on Evolving Graphs: Citation data
2. Inductive Learning on Evolving Graphs: Reddit data
3. Generalization across graphs: Protein-protein interactions

Experiments

Baselines

- Random classifier
- Logistic regression feature-based classifier
- DeepWalk algorithm
- Raw features + DeepWalk embeddings

GraphSAGE variants

- GraphSAGE-GCN
- GraphSAGE-mean
- GraphSAGE- LSTM
- GraphSAGE-pool

Experiment Design

Goals

(1) Verifying the improvement of GraphSAGE over the baseline approaches

(2) Providing a rigorous comparison of the different GraphSAGE aggregator architectures

Experiments – Evolving Information Graphs

Citation Data

Task

Predict paper subject category on a large citation dataset.

Features

- Node degrees.
- Sentence embeddings for abstracts.
- GenSim word2vec 300-dim word vectors.

Data

- From the Thomson Reuters Web of Science Core Collection.
- All papers in 6 biology-related fields for the years 2000-2005.
- 302,424 nodes with and avg. degree of 9.15.
- Training: 2000-2004.
- Test: 2005(30% validation).

Experiments – Evolving Information Graphs

Reddit data

Task

Predict which community different Reddit posts belong to.

Features

- Off the shelf 300-dim word vectors.
- Per post:
  - Concatenate avg. embedding of post title, avg. embeddings of all comments, post score, number of comments.

Data

- Posts connected if the same user comments on both.
- 232,965 posts with an avg. degree of 492.
- Post of 50 subreddits/communities made in Sep/2014.
- Training: First 20 days.
- Test: Remaining days (30% validation).

Experiments – Generalization Across Graphs

PPI – Protein-Protein Interactions

Task

Classify protein roles in PPI graphs.
Each graph from a different human tissue.
Labels: gene ontology sets (121 total)

Features

- Positional gene sets,
- motif gene sets,
- immunological signatures.

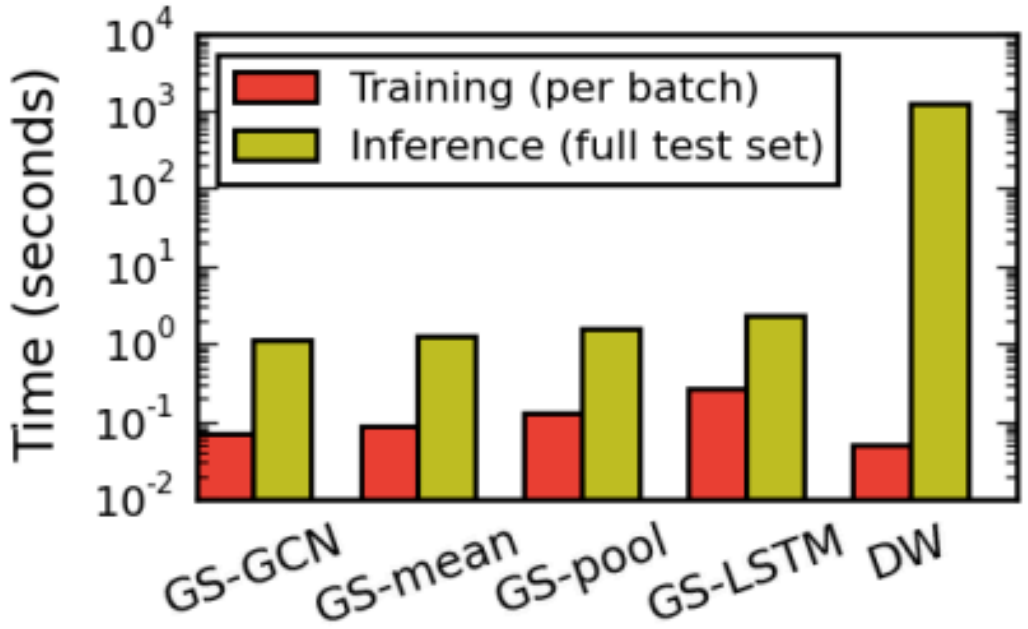
Data

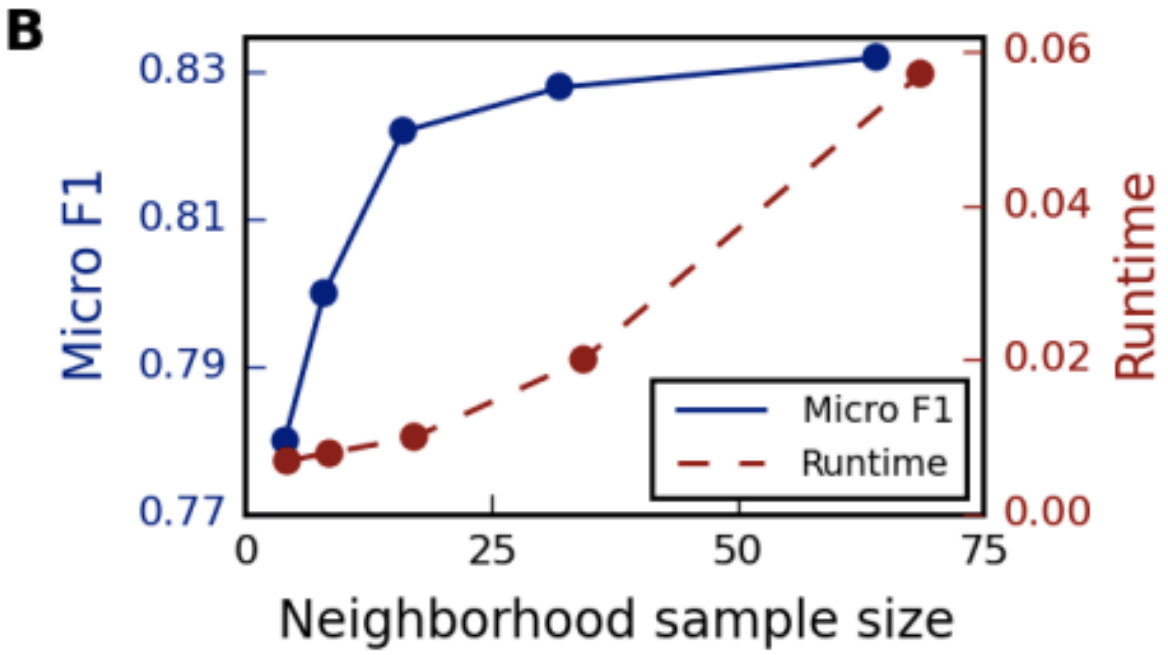
- Avg: 2372 nodes/graph with an avg. degree of 28.8.
- Training: 20 graphs.
- Test: 2 graphs (+ 2 graphs for validation).

Results

	Citation		Reddit		PPI	
Name	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1
Random	0.206	0.206	0.043	0.042	0.396	0.396
Raw features	0.575	0.575	0.585	0.585	0.422	0.422
DeepWalk	0.565	0.565	0.324	0.324	—	—
DeepWalk + features	0.701	0.701	0.691	0.691	—	—
GraphSAGE-GCN	0.742	0.772	**0.908**	0.930	0.465	0.500
GraphSAGE-mean	0.778	0.820	0.897	0.950	0.486	0.598
GraphSAGE-LSTM	0.788	0.832	**0.907**	**0.954**	0.482	**0.612**
GraphSAGE-pool	**0.798**	**0.839**	0.892	0.948	**0.502**	0.600
% gain over feat.	39%	46%	55%	63%	19%	45%

Results


Time (seconds)
10^4
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
Training (per batch)
Inference (full test set)
GS-GCN
GS-mean
GS-pool
GS-LSTM
DW


B
Micro F1
0.83
0.81
0.79
0.77
Runtime
0.06
0.04
0.02
0.00
0
25
50
75
Neighborhood sample size
Micro F1
Runtime

Summary

GraphSAGE (SAmple and aggreGatE)

- Approach that leverages node features in order to learn an embedding function that generalizes to unseen nodes.
- It can also make use of graph structural features and applied to graph without node features.
- Train a set of aggregator functions that learn to aggregate feature information from a node's local neighborhood.
- At test or inference time, generate embeddings for entirely unseen nodes by applying the learned aggregate functions.
- Can be trained without task-specific supervision, and also on a fully supervised manner.

Discussion Questions

1. What could explain that the LSTM aggregator, despite not being inherently symmetric, shows strong performance?
2. What are the ideal characteristics of aggregators and what other potential functions could be considered for this function?